APPENDIX A.IV

Source code file named liboaa.pl.

```
%*******************************************************************************
%   File     : liboaa.pl
%   Primary Authors  : Adam Cheyer, David Martin
%   Purpose : Prolog version of library for the Open Agent Architecture
%   Updated : 12/98
%
%   ------------------------------------------------------------------------
%   Unpublished-rights reserved under the copyright laws of the United States.
%

%
%   Unpublished Copyright (c) 1998, SRI International.
%   "Open Agent Architecture" and "OAA" are Trademarks of SRI International.
%   ------------------------------------------------------------------------
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Note: internal functions use the naming convention oaa_function_name(),
%     while public predicates use oaa_PublicPredicate().
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Version 2.0 (change oaa_version assertion)
%    - corrects FromKS in do_events by changing event format to include this
%      info.
%    - messages are only sent to READY agents.  For previous versions, an
%      agent may be either READY or just OPEN.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Version 2.1 (change oaa_version assertion)
%    - triggers have 2 new arguments, OpMask and Template, and
%      more general semantics.  Backwards compatibility is provided.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Version 3.0 (change oaa_version assertion)
%    - primitives changed to start with oaa_ (and _icl) prefixes
%    - Major restructuring and cleanup, including many new capabilities,
%      for first public release (a.k.a. "OAA 2")
%*******************************************************************************

:- module(oaa,
        [icl_GetParamValue/2,
         icl_GetPermValue/2,
         icl_BasicGoal/1,
         icl_GoalComponents/4,
         icl_ConsistentParams/2,
         icl_BuiltIn/1,
         icl_ConvertSolvables/2,
         oaa_LibraryVersion/1,
         oaa_Register/3,
         oaa_RegisterCallback/2,
         oaa_ResolveVariables/1,
         oaa_Ready/1,
         oaa_MainLoop/1,
         oaa_SetTimeout/1,
         oaa_GetEvent/3,
         oaa_ProcessEvent/2,
         oaa_Interpret/2,
         oaa_DelaySolution/1,
         oaa_ReturnDelayedSolutions/2,
         oaa_AddDelayedContextParams/3,
```

```
        oaa_PostEvent/2,
        oaa_CanSolve/2,
        oaa_Version/3,
        oaa_Ping/3,
        oaa_Declare/5,
        oaa_DeclareData/3,
        oaa_Undeclare/3,
        oaa_Redeclare/3,
        oaa_AddData/2,
        oaa_RemoveData/2,
        oaa_ReplaceData/3,
        oaa_CheckTriggers/3,
        oaa_AddTrigger/4,
        oaa_RemoveTrigger/4,
        oaa_Solve/2,
        oaa_InCache/2,
        oaa_AddToCache/2,
        oaa_ClearCache/0,
        oaa_TraceMsg/2,
        oaa_ComTraceMsg/2,
        oaa_Inform/3,
        oaa_Id/1,
        oaa_Name/1
       ]).


%*****************************************************************************
%* RCS Header and internal version
%*****************************************************************************

% rcs version number
rcsid('$Header: /home/trestle4/OAA/src/V2/prolog/RCS/oaa.pl,v 1.127 1998/12/23
23:14:18 martin Exp cheyer $').

:- op(599,yfx,::).

%*****************************************************************************
% Include files
%*****************************************************************************
:- use_module(library(basics)).
:- use_module(library(read_sent)).
:- use_module(library(lists)).
:- use_module(library(sets)).
:- use_module(library(strings)).
:- use_module(library(files)).
:- use_module(library(environ)).  % read environment vars
:- use_module(library(ctr)).
:- use_module(library(charsio)).  % for sprintf and with_output_to_chars
:- use_module(library(ask)).      % for ask_oneof
:- use_module(library(samsort)).  % for samsort(Ordered,Raw,Sort)
:- use_module(library(date)).     % for now(Time)

:- use_module(library(tcp), [tcp_now/1, tcp_time_plus/3]).


% IMPORTANT: COM module.  We don't want to hard code the name of the
```

```
% file that contains module 'com'.  So, when this file is loaded,
% we first check to see if module 'com' is already present, then
% we check to see if the file containing 'com' has been specified
% on the command line, and if neither of those works, we load the
% default file (./com_tcp).
%
% In the case where the module has already been
% loaded, the following seems like the right thing to do:
%     :- use_module(com, _File, all).
% BUT when compiling, this approach results in "undefined" errors from
% qcon.  Thus, for now, in oaa.pl, we are explicitly using com: with all
% calls to the com module.

:- ( current_predicate(_, com:_) ->
       use_module(com, _File, all)
   | unix(argv(ListOfArgs)), append(_, ['-com', File | _], ListOfArgs) ->
       use_module(File, all)
   | otherwise ->
       use_module(com_tcp, all)
   ).


%*****************************************************************************
% Global variables
%*****************************************************************************
:- dynamic
            oaa_already_loaded/1,    % record if file already loaded
                oaa_solvables/1,         % list of agent capabilities
                oaa_trigger/5,           % a built-in solvable
            oaa_trace/1,             % trace mode: on or off
            oaa_com_trace/1,  % com_trace mode: on or off
            oaa_debug/1,             % debug mode: on or off
            oaa_cache/2,             % cached solutions
            oaa_event_buffer/1,      % buffer of waiting events
            oaa_waiting_for/2,       % used for recursive blocking solve
            oaa_waiting_event/1,     %    problem...
            oaa_timeout/1,           % tcp timeout value (use oaa_SetTimeout)
            oaa_delay_table/5,       % table of delayed solutions
            oaa_delay/2,             % the current goal is delayed
            oaa_data_ref/3,          % bookkeeping for 'data' solvables
            oaa_current_contexts/2, % Solve parameters to be propagated
            oaa_callback/2,          % Record of app-specific callbacks
    % These may appear in setup.pl:
                oaa_host/1,              % for root, my host; otherwise,
                                         % host of my parent
                oaa_port/1.              % ... similarly ...



oaa_LibraryVersion(3.0).

     % solvables shared by all agents
     % Note: all built-in DATA solvables must be declared dynamic to avoid
     % QP warnings and exceptions.
oaa_built_in_solvables([
  % @@DLM: If we do away with TriggerId, we could use param
  % unique_values(true).
```

```
  solvable(oaa_trigger(_TriggerId, _Type, _Condition, _Action, _Params),
                       [type(data)], [write(true)])
]).

% We'll always have exactly one oaa_solvables fact.  Note that application
% code should NOT include a declaration or clause for oaa_solvables/1.
oaa_solvables([]).



%*****************************************************************************
% Initialization and connection functions
%*****************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Register
% purpose: Once a comm link is established, either as a client to a Facilitator
%  or as a server for other agents, oaa_Register will setup and registration
%  information for this agent.
%  inputs:
%   - ConnectionId: the symbolic connection Id (client or server connection)
%   - AgentName: the name of the agent
%   - Solvables: solvable list
% remarks:
%   The following information is stored about the current connection,
%   accessible through com_GetInfo(ConnectionId, Info):
%
%     oaa_name(Name)     : the name of the current agent
%     oaa_id(Id)   : the Id for the agent
%       connection(C)    : system-level communications handle
%                          (e.g., socket number)
%
%   if connecting as client, this is also available:
%     fac_id(Id)   : the Facilitator's Id
%     fac_name(Name)     : the Facilitator's name
%     fac_lang(L) : the Facilitator's language
%     fac_version(V)     : the version of the Facilitator's agent library
%
%   In addition, the following predicates are written to parent Facilitator,
%   or locally if the ConnectionId is a server connection:
%
%     agent_host(Id, Name, Host)
%
%   Solvables are also written using oaa_Declare()
%
%   It is possible for an agent to create both server and client connections:
%   such an agent was classified in OAA 1.0 as an agent of class "node"
%   (as opposed to a pure client "leaf" or pure server "root").
%
% examples:
%   % connecting to a Facilitator
%     MySolvables = [do(something)],
%     com_Connect(parent, ConnectionInfo),
%     oaa_Register(parent, my_agent_name, MySolvables).
%
%   % connecting as a Facilitator
```

```
%     MySolvables = [],
%     com_ListenAt(incoming, ConnectionInfo),
%     oaa_Register(incoming, root, MySolvables).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% For client connecting to Facilitator
oaa_Register(ConnectionId, AgentName, Solvables) :-
      % succeeds only if exists an open client connection for ConnectionId
      %    as created by com_Connect()
      com:com_connection_info(ConnectionId, _Protocol, client, _Info,
connected),

      com:com_AddInfo(ConnectionId, oaa_name(AgentName)),

      % FIXED HACK: default now works thanks to update in com_tcp.pl for
      %    the default mode
      % HACK!!! Why doesn't this work right without it?
      % for some reason, when we send the handshaking info in
      %    default mode (instead of quintus_binary), the facilitator's
      %    tcp_select(VerySmallTimeout, Event) doesn't timeout!!!!
      %    So it keeps hanging until some other event (such as disconnect)
      %    arrives.
      com:com_AddInfo(ConnectionId, format(default)),

      % lookupversion number
      oaa_LibraryVersion(Version),

      %%% handshaking with Facilitator -- exchange information...
      % note: for this first communication, no format is defined for the
      %       connection, so it will be sent using default (ascii) format.
      %       Information coming back from Facilitator will update the
      %       format() field for the connection, improving future
      %       communication.
      com:com_SendData(ConnectionId,
            event(ev_connect([oaa_name(AgentName), agent_language(prolog),
             format(quintus_binary), agent_version(Version)]), [])),

      %% Get the connection acknowledgement:
      %  potential bug: what if selected event is NOT from FacId connection?
      oaa_GetEvent(ConnEvent, _Parms, 0),
      ConnEvent = ev_connected(FacInfoList),
      com:com_AddInfo(ConnectionId, FacInfoList),

      oaa_Id(MyId),

        % write host
        ( environ('HOST', MyHost) ->
           oaa_AddData(agent_host(MyId, AgentName, MyHost), [address(parent)])
        | true),

        % Declare solvables (and post to parent facilitator):
        % Note: OK if Solvables = [].
        oaa_Declare(Solvables, [], [], [if_exists(overwrite)], _).

% For Faciliator serving client agents
oaa_Register(ConnectionId, AgentName, Solvables) :-
```

```
      % succeeds only if exists an open client connection for ConnectionId
      %    as created by com_Connect()
      com:com_connection_info(ConnectionId, _Protocol, server, _Info,
connected),

      AgentId = 0,  % A facilitator's ID is always 0
      com:com_AddInfo(ConnectionId, [oaa_id(AgentId),oaa_name(AgentName)]),

      % The fac. records its own agent_data in the same way as its clients'.
        % Note that we can't call oaa_add_data_local until after the solvables
        % have been declared, and we can't declare solvables until we're
        % open - so we have to bootstrap this assertion:
        oaa_assertz(agent_data(AgentId, open, [], AgentName), AgentId, _),

        % Note: OK if Solvables = [].
        oaa_Declare(Solvables, [], [], [if_exists(overwrite)], _),

        % write host
        ( environ('HOST', MyHost) ->
          oaa_add_data_local(agent_host(AgentId, AgentName, MyHost),[])
        | true).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% name:    oaa_ResolveVariables(+VariableList)
% purpose: Tries to instantiate the arguments by looking in the command
%        line arguments, environment variables, and setup files
%  inputs:
%   - VarList:    A list of lists: the first sublist that completely resolves
%        provides the value for oaa_ResolveVariables.
% remarks:
%    sublists may contain elements in the following format:
%       env(EnvVar, Val)       : looks for "EnvVar" in environment vars
%     env_int(EnvVar, Val)     : Returns value for EnvVar as an integer
%     cmd(CmdVar, Val)   : looks for "CmdVar <Val>" on command line
%     setup(SVar, Val)   : reads SVar from setup file
% example:
%    resolves host and port by searching first commandline, then environment
%    variables, finally reads setup file.
%
%     oaa_ResolveVariables([
%         [cmd('-oaa_host',Host), cmd('-oaa_port', Port)],
%       [env('OAA_HOST', Host), env_int('OAA_PORT', Port)],
%       [setup(oaa_host, Host), setup(oaa_port, Port)]
%     ])
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ResolveVariables([VarList|_]) :-
      oaa_resolve_variables(VarList), !.
oaa_ResolveVariables([_VarList|Rest]) :-
      oaa_ResolveVariables(Rest).

oaa_resolve_variables([]).

oaa_resolve_variables([env_int(EnvVar, Val)|Rest]) :- !,
```

```
      environ(EnvVar, EnvAtom),
      name(EnvAtom, EnvChars),
      number_chars(Val, EnvChars),
      oaa_resolve_variables(Rest).

oaa_resolve_variables([env(EnvVar, Val)|Rest]) :- !,
      environ(EnvVar, Val),
      oaa_resolve_variables(Rest).

oaa_resolve_variables([cmd(CmdVar, Val)|Rest]) :- !,
      % get command line arguments
      unix(argv(ListOfArgs)),
      append(_, [CmdVar, Val|_], ListOfArgs),
      oaa_resolve_variables(Rest).

oaa_resolve_variables([setup(SVar, Val)|Rest]) :- !,
      % read setup file to load all values
      oaa_read_setup_file,
      Pred =.. [SVar, Val],
        on_exception(_, Pred, fail),
      oaa_resolve_variables(Rest).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_read_setup_file
% purpose: Finds and loads setup file
% remarks:
%    Always succeeds.
%    The search path for 'setup.pl' is as follows:
%      1. Current directory
%      2. Home directory for user
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_read_setup_file :-
      oaa_already_loaded(setup), !.
oaa_read_setup_file :-
      ( absolute_file_name('setup.pl', LocalSetupFile),
        can_open_file(LocalSetupFile, read, fail) ->
          SetupFile = LocalSetupFile
      | absolute_file_name('~/setup.pl', UserSetupFile),
          can_open_file(UserSetupFile, read, fail) ->
             SetupFile = UserSetupFile
      ),

      (ground(SetupFile) ->
         format('Loading OAA setup file:~n  ~w~n', [SetupFile]),
         ( oaa_consult(SetupFile, _) ->
             assert(oaa_already_loaded(setup))
         | otherwise ->
             format('~w: A problem was encountered in loading the setup file~n',
                  ['WARNING'])
         )
      | true).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Ready
% purpose: Changes the agent's 'open' status to 'ready', indicating that the
%        agent is now ready to receive messages.
% remarks:
%    if requested, prints 'Ready' to standard out.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Ready(ShouldPrint) :-

    % replaces 'open' status with 'ready'.
    ((\+ oaa_class(root), oaa_Name(MySymbolicName)) ->
      oaa_PostEvent(ev_ready(MySymbolicName), [])
    | true),

    % if ShouldPrint, print ready
    (on_exception(_,ShouldPrint,fail) ->
       format('Ready.~n', [])
    | true).



%*************************************************************************
% Classifying and Manipulating ICL expressions
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_BuiltIn(+Goal).
% purpose: Test whether an expression is an ICL built-in goal.
% remarks:
%     - icl_BuiltIn differs significantly from the Quintus Prolog predicate
%       built_in, in that here we do not include basic constructors such
%       as ',' and ';'.
%     - oaa_Interpret/2 must be defined for every goal for which
%       icl_BuiltIn succeeds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_BuiltIn((_A = _B)).
icl_BuiltIn((_A == _B)).
icl_BuiltIn((_A \== _B)).
icl_BuiltIn((_A =< _B)).
icl_BuiltIn((_A >= _B)).
icl_BuiltIn((_A < _B)).
icl_BuiltIn((_A > _B)).
icl_BuiltIn(member(_,_)).
icl_BuiltIn(memberchk(_,_)).
icl_BuiltIn(findall(_,_,_)).
icl_BuiltIn(icl_ConsistentParams(_,_)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_BasicGoal(+Goal).
% purpose: Test whether an expression is an ICL basic (non-compound) goal;
%          that is, just a functor with 0 or more arguments.
% remarks:
%     - Basic goals include built-in's as well as solvables.
%     - This is a syntactic test; that is, we're not checking whether the
%       Goal is a declared solvable.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_BasicGoal(Goal) :-
    var(Goal), !, fail.
icl_BasicGoal(Goal) :-
    is_list(Goal), !, fail.
icl_BasicGoal(Goal) :-
    icl_compound_goal(Goal), !, fail.
icl_BasicGoal(Goal) :-
    icl_BuiltIn(Goal),
    !.
icl_BasicGoal(Goal) :-
    Goal =.. [Functor | _],
    atom(Functor).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_compound_goal(+Goal).
% purpose: Test whether an expression is an ICL compound goal.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_compound_goal(_X:_Y).
icl_compound_goal(_X::_Y).
icl_compound_goal((\+ _P)).
icl_compound_goal((_P -> _Q ; _R)).
icl_compound_goal((_P -> _Q)).
icl_compound_goal((_X, _Y)).
icl_compound_goal((_X ; _Y)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_GoalComponents(+ICLGoal, -A, -G, -P).
%          icl_GoalComponents(-ICLGoal, +A, +G, +P).
%          icl_GoalComponents(+ICLGoal, +A, +G, +P).
% purpose: Assemble, disassemble, or match against the top-level components
%          of an ICL goal.
% remarks:
%     - The top-level structure of an ICL goal is Address:Goal::Params,
%       with Address and Params BOTH OPTIONAL.  Thus, every ICL goal
%       either explicitly or implicitly includes all three components.
%     - This may be used with any ICL goal, basic or compound.
%     - When P is missing, its value is returned or matched as [].  When A is
%       missing, its value is returned or matched as 'unknown'.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The first 4 clauses handled all cases where the ICL Goal is bound;
% the remainder handle those where it is a var.
icl_GoalComponents(A:G::P, Address, Goal, Params) :-
    \+ var(A), \+ var(G), \+ var(P),
    !,
    Address = A, Goal = G, Params = P.
icl_GoalComponents(A:G, Address, Goal, Params) :-
    \+ var(A), \+ var(G),
    !,
    Address = A, Goal = G, Params = [].
icl_GoalComponents(G::P, Address, Goal, Params) :-
    \+ var(G), \+ var(P),
    !,
    Address = unknown, Goal = G, Params = P.
```

```
icl_GoalComponents(G, Address, Goal, Params) :-
    \+ var(G),
    !,
    Address = unknown, Goal = G, Params = [].
icl_GoalComponents(Goal, unknown, Goal, []) :-
    !.
icl_GoalComponents(Address:Goal, Address, Goal, []) :-
    !.
icl_GoalComponents(Goal::Params, unknown, Goal, Params) :-
    !.
icl_GoalComponents(Address:Goal::Params, Address, Goal, Params) :-
    !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Permissions and parameter lists
%
% These procedures are used in processing solvables permissions, and
% parameter lists of all kinds (including those used with solvables,
% those contained in events, and those used in calls to various
% library procedures).
%
% All permissions and many parameters have default values.
%
% Permissions and parameters lists have a standard form, as defined by
% the predicates below.  To save bandwidth and promote readability, a
% "perm" or "param" list in standard form OMITS default values.  For
% easier processing (e.g., comparing/merging param lists), boolean
% params in standard form always include a single argument 'true' or
% 'false'.
%
% In definitions of solvables and calls to documented library
% procedures, it's OK to include default params in a Params list, if
% desired.  For boolean params, when the intended value is 'true', it's
% OK just to specify the functor, for example, instead of
% cache(true), it's OK just to include 'cache'.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% icl_standardize_perms(+Perms, +KeepDefaults, -Standardized).

icl_standardize_perms([], _KeepDefaults, []).
icl_standardize_perms([Perm | Perms], KeepDefaults, [SPerm | SPerms]) :-
    icl_perm_standard_form(Perm, SPerm),
    ( KeepDefaults ; (\+ icl_perm_default(SPerm)) ),
    !,
    icl_standardize_perms(Perms, KeepDefaults, SPerms).
icl_standardize_perms([_Perm | Perms], KeepDefaults, SPerms) :-
    icl_standardize_perms(Perms, KeepDefaults, SPerms).

icl_perm_standard_form(Perm, SPerm) :-
    atom(Perm),
    !,
    SPerm =.. [Perm, true].
icl_perm_standard_form(Perm, Perm).

icl_perm_default(call(true)).
```

```
icl_perm_default(read(false)).
icl_perm_default(write(false)).

% icl_standardize_params(+Params, +KeepDefaults, -Standardized).
%
% Normally there's no need to keep the default value of a param,
% but there are exceptional situations.  If KeepDefaults is true,
% default values are kept.

icl_standardize_params([], _, []).
icl_standardize_params([Param | Rest], KeepDefaults, AllStandardized) :-
    icl_param_standard_form(Param, FullStandardized),
    ( KeepDefaults ->
      Standardized = FullStandardized
    | otherwise ->
      icl_remove_default_params(FullStandardized, Standardized)
    ),
    icl_standardize_params(Rest, KeepDefaults, RestStandardized),
    append(Standardized, RestStandardized, AllStandardized).

% icl_param_standard_form(+Param, -StandardParams).
%
% Maps from an element of a parameter list to a list of elements
% in standardized form.  The parameter list element can be from
% any context (from a call to Solve, AddTrigger, AddData, etc.).

icl_param_standard_form(reply(false), [reply(none)]) :-
    !.
    % broadcast has been retained, as a synonym for reply(none):
icl_param_standard_form(broadcast, [reply(none)]) :-
    !.
icl_param_standard_form(broadcast(true),  [reply(none)]) :-
    !.
icl_param_standard_form(broadcast(false),  [reply(true)]) :-
    !.
icl_param_standard_form(address(Addr),  [address(SAddr)]) :-
    !,
    icl_standardize_address(Addr, SAddr).
icl_param_standard_form(strategy(query), [parallel_ok(true)]) :-
    !.
icl_param_standard_form(strategy(action),
                [parallel_ok(false), solution_limit(1)]) :-
    !.
icl_param_standard_form(strategy(inform),
                [parallel_ok(true), reply(none)]) :-
    !.
icl_param_standard_form(callback(Mod:Proc), [callback(Mod:Proc)]) :-
    !.
icl_param_standard_form(callback(Proc), [callback(user:Proc)]) :-
    !.
icl_param_standard_form(Param,  [SParam]) :-
    atom(Param),
    !,
    SParam =.. [Param, true].
icl_param_standard_form(Param, [Param]).

icl_param_default(from(unknown)).
```

```
icl_param_default(priority(5)).
icl_param_default(utility(5)).
icl_param_default(if_exists(append)).
icl_param_default(type(procedure)).
icl_param_default(private(false)).
icl_param_default(single_value(false)).
icl_param_default(unique_values(false)).
icl_param_default(rules_ok(false)).
icl_param_default(bookkeeping(true)).
icl_param_default(persistent(false)).
icl_param_default(at_beginning(false)).
icl_param_default(do_all(false)).
icl_param_default(reflexive(true)).
icl_param_default(parallel_ok(true)).
icl_param_default(reply(true)).
icl_param_default(block(true)).
icl_param_default(cache(false)).
icl_param_default(flush_events(false)).
icl_param_default(recurrence(when)).

icl_remove_default_params([], []).
icl_remove_default_params([Param | Rest], Removed) :-
    icl_param_default(Param),
    !,
    icl_remove_default_params(Rest, Removed).
icl_remove_default_params([Param | Rest], [Param | Removed]) :-
    icl_remove_default_params(Rest, Removed).

% icl_GetParamValue(+Param, +ParamList).
%
% Param must have a functor, but its argument(s) can be either ground
% or variables.  E.g., persistent(X).
%
% To get or test the value of a parameter that has a default, it is
% best to call icl_GetParamValue.  For a parameter that has no default,
% you can use icl_GetParamValue OR memberchk.

icl_GetParamValue(Param, ParamList) :-
    predicate_skeleton(Param, Skel),
    memberchk(Skel, ParamList),
    !,
    Skel = Param.
icl_GetParamValue(Param, _ParamList) :-
    predicate_skeleton(Param, Skel),
    icl_param_default(Skel),
    !,
    Skel = Param.

icl_GetPermValue(Perm, PermList) :-
    predicate_skeleton(Perm, Skel),
    memberchk(Skel, PermList),
    !,
    Skel = Perm.
icl_GetPermValue(Perm, _PermList) :-
    predicate_skeleton(Perm, Skel),
    icl_perm_default(Skel),
    !,
```

```
    Skel = Perm.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_ConsistentParams(+Test, +ParamList)
% purpose: Often used in solvable declarations to filter on a certain
%        condition.
% definition:
%          Test a param list: if one or more values are given in a parameter
%        list for parameter ParamName, then ParamValue must be defined as
%        one of the values to succeed.  If ParamValue is NOT defined, then
%        icl_ConsistentParams succeeds.
% example:
%    A natural language parser agent can only handle English definitions:
%
%        convert(nl, icl,Input,Params,Output) :-
%           icl_ConsistentParams(language(english),Params).
%
%    if "language(english)" is defined in parameter list of a solve request,
%       the nl agent will receive the request.
%    if "language(spanish)" is defined in the parameter list, the nl agent
%       WILL NOT receive the request.
%    if no language parameter is specified, the request WILL be sent
%    if "language(X)" is specified, the request WILL be sent to the nl agent
% remarks:
%    - Test may contain either a single predicate or a list of test predicates,
%      in which case icl_ConsistentParams will execute all consistency tests.
%    - Interesting note: icl_ConsistentParams() checks consistency as a
%      relation between the two arguments, so it doesn't matter which argument
%      specifies the test list and which the parameters to test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

icl_ConsistentParams(_TestList, []) :- !.
icl_ConsistentParams([], _ParamList) :- !.
icl_ConsistentParams([Test|RTest], [P1|RParams]) :- !,
   ParamList = [P1|RParams],
   predicate_skeleton(Test, TestWithVars),
   (memberchk(TestWithVars, ParamList) ->
      memberchk(Test, ParamList)
   | true),
   icl_ConsistentParams(RTest, ParamList).
% either Test or Params is NOT a list
icl_ConsistentParams(Test, Param) :-
      (Test = [_|_] ->
         NewTest = Test
      |  NewTest = [Test]),
      (Param = [_|_] ->
         NewParam = Param
      |  NewParam = [Param]),
      icl_ConsistentParams(NewTest, NewParam).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Agent identity and addressing
%
% Every agent (including facilitators) has a symbolic name, a full address,
```

```
% and a local address (or "local ID").  A full address has the form:
%     addr(tcp(Host,Port))              for a facilitator (if TCP is protocol)
%     addr(tcp(Host,Port), LocalID)     for a client agent.
%
% Even though it doesn't appear in the full address, a facilitator also
% has a local ID, for consistency and convenient reference. The
% local ID of a client agent is assigned to it by its facilitator.
% This, and the facilitator's local ID, are passed to the client at
% connection time.
%
% Full addresses are globally unique, and local addresses are unique with
% respect to a facilitator.  Symbolic names are NOT unique in any sense.
%
% The local ID happens to be an integer, but developers should not rely
% on this.
%
% When specifying addresses, in address/1 params for calls to
% oaa_AddData, oaa_Solve, etc., either names or addresses may be used.
% In addition, for convenience, reserved terms 'self', 'parent', and
% 'facilitator' may also be used.
%
% More precisely, the address parameter may contain any of the following:
% a full address; a local ID (when the addressee is known to be either
% the facilitator or a peer client); a name, enclosed in the name/1 functor;
% 'self'; 'parent'; or 'facilitator'.  ('parent' and 'facilitator are
% synonymous.)
%
% Address parameters are standardized as follows: A full address for the
% local facilitator or a peer client is changed to the local ID; all
% other full addresses are left as is.  Names are left as is.  'self',
% 'parent', and 'facilitator' are changed to the appropriate local ID.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % This can only be used AFTER oaa_SetupCommunication has been called,
    % because of the reliance here on com:com_connection_info/5.
icl_standardize_address(Addr, SAddr) :-
    \+ is_list(Addr),
    !,
    icl_standardize_address([Addr], SAddr).
icl_standardize_address([], []).
icl_standardize_address([Addr | Addrs], [SAddr | SAddrs]) :-
    icl_standardize_addressee(Addr, SAddr),
    !,
    icl_standardize_address(Addrs, SAddrs).
icl_standardize_address([_Addr | Addrs], SAddrs) :-
    icl_standardize_address(Addrs, SAddrs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

icl_standardize_addressee(addr(Addr), ParentId) :-
    com:com_GetInfo(parent, addr(Addr)),
    com:com_GetInfo(parent, fac_id(ParentId)),
    !.
icl_standardize_addressee(addr(Addr), addr(Addr)) :-
    !.
icl_standardize_addressee(addr(Addr, LID), LID) :-
```

```
    com:com_GetInfo(parent, addr(Addr)),
    !.
icl_standardize_addressee(addr(Addr, LID), LID) :-
    com:com_GetInfo(incoming, addr(Addr)),
    !.
icl_standardize_addressee(addr(Addr, LID), addr(Addr, LID)) :-
    !.
icl_standardize_addressee(name(Name), name(Name)) :-
    !,
    icl_name(Name).
icl_standardize_addressee(Name, name(Name)) :-
    icl_name(Name),
    !,
    format('~w (~w): addressee name, in address/1 param, should be specified
as:~n  name(~w)~n',
         ['WARNING', 'liboaa.pl', Name]).
icl_standardize_addressee(Id, TrueId) :-
    icl_true_id(Id, TrueId),
    !.
icl_standardize_addressee(Whatever, _) :-
    format('~w (~w): Illegal addressee, in address/1 param, discarded:~n  ~w~n',
         ['WARNING', 'liboaa.pl', Whatever]),
    fail.

icl_true_id(self, Me) :-
    !,
    oaa_Id(Me).
icl_true_id(parent, Parent) :-
    !,
    com:com_GetInfo(parent, fac_id(Parent)).
icl_true_id(facilitator, Parent) :-
    !,
    com:com_GetInfo(parent, fac_id(Parent)).
icl_true_id(Id, Id) :-
    icl_id(Id).

icl_id(Num) :-
    integer(Num),
    Num >= 0.

icl_name(self) :-
    !, fail.
icl_name(parent) :-
    !, fail.
icl_name(facilitator) :-
    !, fail.
icl_name(Atom) :-
    atom(Atom).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_ConvertSolvables(+ShorthandSolvables, -StandardSolvables).
%          icl_ConvertSolvables(-ShorthandSolvables, +StandardSolvables).
%
% purpose: Convert between shorthand and standard forms of solvables list.
% remarks:
%     - In the standard form, each element is a term solvable(Goal,
```

```
%       Params, Permissions), with Permissions and Params both lists.
%     In the Permissions and Params lists, values appear only when they
%       are OTHER than the default.
%     - In the shorthand form, each element can be solvable/3, as above,
%       or solvable(Goal, Params), or solvable(Goal), or just Goal.
%     - Note that "shorthand" means "anything goes" - so shorthand
%       solvables are a superset of standard solvables.
%     - Permissions (defaults in square brackets):
%         call(T_F) [true], read(T_F) [false], write(T_F) [false]
%     - Params (defaults in square brackets):
%         type(Data_Procedure) [procedure],
%         callback(Functor) [no default]
%         utility(N) [5]
%         synonym(SynonymHead, RealHead) [none]
%         rules_ok(T_F) [false],
%         single_value(T_F) [false],
%       unique_values(T_F) [false],
%         private(T_F) [false]
%         bookkeeping(T_F) [true]
%         persistent(T_F) [false]
%     - Refer to Agent Library Reference Manual for details on Permissions
%       and Params.
%     - (@@DLM) This might be the place to check the validity of solvables,
%       such as using only built-ins in tests.  Also, check for dependencies
%       between solvables; e.g., when persistent(false) is there,
%       bookkeeping(true) must also be there.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
icl_ConvertSolvables(ShorthandSolvables, StandardSolvables) :-
    var(StandardSolvables),
    !,
    icl_standardize_solvables(ShorthandSolvables, StandardSolvables).
icl_ConvertSolvables(ShorthandSolvables, StandardSolvables) :-
    icl_readable_solvables(StandardSolvables, ShorthandSolvables).

    % icl_standardize_solvables(+ShorthandSolvables,
    %                           -StandardSolvables).
icl_standardize_solvables([], []).
icl_standardize_solvables([Shorthand | RestSH], [Standard | RestStan]) :-
    icl_standardize_solvable(Shorthand, Standard),
    icl_standardize_solvables(RestSH, RestStan).

    % icl_standardize_solvable(+Shorthand, -Standard).
icl_standardize_solvable(solvable((Goal :- Test), Params, Perms), Standard) :-
    !,
    append([test(Test)], Params, NewParams),
    icl_standardize_solvable(solvable(Goal, NewParams, Perms), Standard).
icl_standardize_solvable(solvable((Goal :- Test), Params), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test) | Params], []),
                        Standard).
icl_standardize_solvable(solvable((Goal :- Test)), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test)], []), Standard).
icl_standardize_solvable((Goal :- Test), Standard) :-
    !,
    icl_standardize_solvable(solvable(Goal, [test(Test)], []), Standard).
```

```
icl_standardize_solvable(solvable(Goal, Params, Perms),
                  solvable(Goal, NewParams, NewPerms)) :-
    !,
    icl_standardize_params(Params, false, NewParams),
    icl_standardize_perms(Perms, false, NewPerms).
icl_standardize_solvable(solvable(Goal, Params),
                  solvable(Goal, NewParams, [])) :-
    !,
    icl_standardize_params(Params, false, NewParams).
icl_standardize_solvable(solvable(Goal), solvable(Goal, [], [])) :- !.
icl_standardize_solvable(Goal, solvable(Goal, [], [])) :- !.

    % icl_readable_solvables(+StandardSolvables,
    %                                  -ShorthandSolvables).
    % This is provided for use in "pretty-printing" solvables, in trace
    % messages, etc.
icl_readable_solvables([], []).
icl_readable_solvables([Standard | RestStan], [Shorthand | RestSh]) :-
    icl_readable_solvable(Standard, Shorthand),
    icl_readable_solvables(RestStan, RestSh).

    % icl_readable_solvable(+Standard, -Shorthand).
icl_readable_solvable(solvable(Goal, [], []), Goal) :- !.
icl_readable_solvable(solvable(Goal, Params, []), solvable(Goal, Params)) :- !.
icl_readable_solvable(solvable(Goal, Params, Perms),
                  solvable(Goal, Params, Perms)) :- !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    icl_minimally_instantiate_solvables(+ShorthandSolvables,
%                                              -MinimalSolvables).
% purpose: Convert from shorthand (or standard form) to minimally instantiated
%          solvables list.
% remarks: - This is special-purpose. It's used to massage a list of solvables
%            that are to be UNdeclared, to make sure each of them will unify
%            with some existing solvable.  Perms and Params are completely
%            ignored in the unification; only the Goal is relevant.  So each
%            minimally instantiated solvable is simply solvable(Goal, _, _).
%          - Note that "shorthand" means "anything goes" - so shorthand
%            solvables are a superset of standard solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % icl_minimally_instantiate_solvables(+ShorthandSolvables,
    %                                     -Solvables).
icl_minimally_instantiate_solvables([], []).
icl_minimally_instantiate_solvables([Shorthand | RestSH],
                            [Minimal | RestMin]) :-
    icl_minimally_instantiate_solvable(Shorthand, Minimal),
    icl_minimally_instantiate_solvables(RestSH, RestMin).

    % icl_minimally_instantiate_solvable(+Shorthand, -Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test), Params, Perms),
                           Minimal) :-
    !,
    icl_minimally_instantiate_solvable(solvable(Goal, Params, Perms),
                               Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test), Params),
                           Minimal) :-
```

```
    !,
    icl_minimally_instantiate_solvable(solvable(Goal, Params, []), Minimal).
icl_minimally_instantiate_solvable(solvable((Goal :- _Test)), Minimal) :-
    !,
    icl_minimally_instantiate_solvable(solvable(Goal, [], []), Minimal).
icl_minimally_instantiate_solvable((Goal :- _Test), Minimal) :-
    !,
    icl_minimally_instantiate_solvable(solvable(Goal, [], []), Minimal).
icl_minimally_instantiate_solvable(solvable(Goal, _Params, _Perms),
                  solvable(Goal, _, _)) :-
    !.
icl_minimally_instantiate_solvable(solvable(Goal, _Params),
                  solvable(Goal, _, _)) :-
    !.
icl_minimally_instantiate_solvable(solvable(Goal), solvable(Goal, _, _)) :- !.
icl_minimally_instantiate_solvable(Goal, solvable(Goal, _, _)) :- !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_goal_matches_solvables(+Goal, +Solvables,
%                                     -RealGoal, -MatchedSolvable).
% purpose: Determine whether a call to Goal is handled by the agent with
%          these Solvables.
% arguments:
%     - Goal must be non-compound (basic) to match: no address, no params,
%       no subgoals.
%     - Solvables must be in standard form.
%     - RealGoal is what should actually be called, after taking synonyms
%       into account.
%     - MatchedSolvable is the solvable record corresponding to RealGoal.
% remarks:
%   - A solvable's params may contain a single test, but it can
%       be compound:
%       solvable(g(X), [test((X > 1,X < 10))], [...]).
%       Tests should contain only prolog builtins.
%   -  Any solvable can be a synonym of another solvable (including a
%       synonym of a synonym), but eventually there must be a non-synonym
%       solvable.  Synonyms must be used with care.  If predicate A
%       is synonymed to predicate B, there must be a solvable for clause B,
%       for A to be usable.
%   -  When a predicate A is synonymed to predicate B, all other params
%       and all permissions associated with A are ignored.
%   -  Uses would_unify (and \+ \+) so that any variables in the goal are
%       not bound by the solvable, thereby unnecessarily constraining query
%       I forget why: I think it was because we had some problems
%       matching solutions coming back.  However, this has an unusual
%       side effect: if your solvable is t(6) and your query is t(X),
%       the query arrives at the agent as t(X), not t(6), which might
%       be unexpected.  Look into this more someday...
%   -  However, when Goal is a synonym, variables in the synonym param DO
%      get unified correctly.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_goal_matches_solvables(Goal, Solvables, RealGoal, RealMatched) :-
    oaa_built_in_solvables(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_goal_in_solvables(Goal, AllSolvables, Matched),
    Matched = solvable(_, Params, _),
```

```
    % See if Goal is a synonym predicate
    ( icl_GetParamValue(synonym(Goal, SynGoal), Params) ->
        oaa_goal_matches_solvables(SynGoal, Solvables, RealGoal, RealMatched)
    | otherwise ->
        RealGoal = Goal,
      RealMatched = Matched
    ),
    !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_goal_in_solvables(+Goal, +Solvables, -MatchedSolvable).
% purpose: Determine whether a call to Goal is handled by the agent with
%          these Solvables.
% purpose: Determine whether Goal appears in Solvables, with
%          appropriate Params and Perms for it to be called.
% arguments:
%     - Goal must be non-compound (basic) to match: no address, no params,
%       no subgoals.
%     - Solvables must be in standard form.
% remarks:
%   - Should not be called directly; only by oaa_goal_matches_solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_goal_in_solvables(Goal, [solvable(G1,Params,Perms) | _Rest],
                  solvable(G1,Params,Perms)) :-
      would_unify(Goal, G1),
        icl_GetParamValue(synonym(Goal, _RealGoal), Params),
      !.
oaa_goal_in_solvables(Goal, [solvable(G1,Params,Perms) | _Rest],
                  solvable(G1,Params,Perms)) :-
      would_unify(Goal, G1),
      icl_GetPermValue(call(true), Perms),
      ( icl_GetParamValue(test(T), Params) ->
          \+ \+ oaa_Interpret((Goal = G1, T), [])
      | otherwise ->
          true
      ),
      !.
oaa_goal_in_solvables(Goal, [_|Rest], Matched) :-
      oaa_goal_in_solvables(Goal, Rest, Matched).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_data_matches_solvables(+Clause, +Solvables, +Perm
%                                     -RealClause, -MatchedSolvable).
% purpose: Determine whether Clause can be read or written by the agent with
%          these Solvables, and return the "real" form of the clause that
%          takes synonyms into account.
% arguments:
%     - Clause must be non-compound (basic) to match: no address, no params,
%       no subClauses.
%     - Solvables must be in standard form.
%     _ Perm is 'read' or 'write'.
%     - RealClause is what should actually be used (asserted, retracted,
%       replaced).
%     - MatchedSolvable is the solvable record corresponding to RealClause.
% remarks:
```

```
%     "Writing" means making an assertion.
%     "Reading" is different than "calling".  "Reading" is retrieving the
%       definition clauses of a predicate (including the bodies, if any).
%       Reading is not currently supported by any library procedures.
%     Any solvable can be a synonym of another solvable (including a
%       synonym of a synonym), but eventually there must be a non-synonym
%       solvable.  Synonyms must be used with care.  If predicate A
%       is synonymed to predicate B, there must be a solvable for clause B,
%       for A to be usable.
%     When a predicate A is synonymed to predicate B, all other params
%       and all permissions associated with A are ignored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_matches_solvables(Clause, Solvables, Perm, RealClause, RealMatched) :-
    oaa_built_in_solvables(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_data_in_solvables(Clause, AllSolvables, Perm, Matched),
    Matched = solvable(_, Params, _),
    ( Clause = (Head :- Body) ->
      true
    | otherwise ->
      Head = Clause
    ),
    % See if Clause is a synonym predicate
    ( icl_GetParamValue(synonym(Head, SynHead), Params) ->
        ( Clause = (Head :- Body) ->
          SynClause = (SynHead :- Body)
      | otherwise ->
          SynClause = SynHead
      ),
        oaa_data_matches_solvables(SynClause, Solvables, Perm,
                                   RealClause, RealMatched)
    | otherwise ->
        RealClause = Clause,
      RealMatched = Matched
    ),
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_data_in_solvables(+Clause, +Solvables, +Perm, -MatchedSolvable).
% purpose: Determine whether (the Head of) Clause appears in Solvables, with
%          appropriate Params and Perms for it to be read or written.
% arguments:
%     - Clause must be non-compound (basic) to match: no address, no params,
%       no subClauses.
%     - Solvables must be in standard form.
% remarks:
%     - Should not be called directly; only by oaa_data_matches_solvables.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_in_solvables(Clause, [solvable(G1,Params,Perms) | _Rest], _Perm,
                  solvable(G1,Params,Perms) ) :-
        ( Clause = (Head :- _Body) ->
            true
      | otherwise ->
          Head = Clause
      ),
      would_unify(Head, G1),
        icl_GetParamValue(synonym(Head, _RealHead), Params),
```

```
      % @@DLM: OK, so it's a synonym, but shouldn't we check
      % the permissions and type(data) for the referenced solvable?
      !.
oaa_data_in_solvables(Clause, [solvable(G1,Params,Perms) | _Rest], Perm,
                  solvable(G1,Params,Perms) ) :-
        icl_GetParamValue(type(data), Params),
        ( Clause = (Head :- _Body) ->
            icl_GetParamValue( rules_ok(true), Params)
      | otherwise ->
          Head = Clause
      ),
      would_unify(Head, G1),
      ( Perm == write ->
          icl_GetPermValue(write(true), Perms)
      | otherwise ->
          icl_GetPermValue(call(true), Perms)
      ),
      !.
oaa_data_in_solvables(Clause, [_|Rest], Perm, Matched) :-
      oaa_data_in_solvables(Clause, Rest, Perm, Matched).



%*************************************************************************
% Retrieving and managing events
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_MainLoop
% purpose: The main event loop for the application.
%          Reads an event, executes (interprets) it,
%        checks on_receive triggers for the event,
%        checks any application-dependent triggers,
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_MainLoop(ShouldPrint) :-

    oaa_Ready(ShouldPrint),

    repeat,
       oaa_GetEvent(Event, Params, 0),
       oaa_ProcessEvent(Event, Params),
    fail.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_ProcessEvent
% purpose: Interprets an incoming event
%    - For a timeout, checks task triggers and calls user's idle procedure
%    - Otherwise, oaa_Interprets the event, checks on_receive comm
%      triggers, and then checks task triggers.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ProcessEvent(timeout, _Params) :- !,
      oaa_CheckTriggers(task, _, _), !,
      oaa_call_callback(app_idle, _, []).
oaa_ProcessEvent(Event, Params) :-
```

```
        ( oaa_Interpret(Event, Params) -> true | true ),
        oaa_CheckTriggers(task, _, _), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_SetTimeout
% purpose: Sets the timeout value used by oaa_GetEvent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_SetTimeout(NSecs) :-
      % Make sure NSecs is valid number
      number(NSecs),
      (NSecs < 0 ->
         TimeOut = 0
      |  TimeOut = NSecs),

      oaa_TraceMsg('~nSetting event timeout to ''~q''.~n', [TimeOut]),
      on_exception(_,retractall(oaa_timeout(_)), true),
      assert(oaa_timeout(TimeOut)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_GetEvent
% purpose: Return the next event to execute
% remarks:
%    - if a oaa_timeout(Secs) is set to a positive real number by
%      oaa_SetTimeout, wait Secs for an event.
%      If none arrives in this time, return Event = `timeout'
%    - Reads ALL events available on communication stream, sorts the events
%      according to priority, chooses the next event to execute,
%      and then saves the rest for next time oaa_GetEvent is called.
%    - The communication stream is read every time oaa_GetEvent is called, even
%      if there are already saved events (a new one might have a higher
%      priority!)
%    - If saved events exist, return immediately (timeout not considered).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_GetEvent(Event, Params, LowestPriority) :-
      % see if previously saved events to process
      ( retract(oaa_event_buffer(SavedEvents)) ->
          true
      | otherwise ->
          SavedEvents = []
      ),

      % If at least one event can be found with an appropriate priority
      %    from among the saved events, no timeout needed -- flush tcp
      %    buffer, and read_all available
      (oaa_choose_event(LowestPriority, SavedEvents, _OneEvent, _Remainder) ->
         TimeoutSecs = 0.01
      |
         on_exception(_,oaa_timeout(TimeoutSecs),TimeoutSecs=0)
      |
         TimeoutSecs=0
      ),

      oaa_read_all_events(TimeoutSecs, MoreEvents, FlushPriority),

      % if one of the new events has a flush in it, see if it
```

```
      %    flushes any of the saved events
      % note: MoreEvents have already been flushed by FlushPriority
      oaa_flush_events(SavedEvents, FlushPriority, RemainingSavedEvents),

      % These are the events we've read so far and haven't executed yet...
      append(RemainingSavedEvents, MoreEvents, EventList),

      (oaa_sort_and_get_event(EventList, LowestPriority, Event, Params) ->
         % we are able to find an appropriate event from list
         % The event will be returned, so fire triggers on it
           oaa_CheckTriggers(comm, event(Event, Params), receive)
      |
         % no good event found, return timeout
         Event = timeout,
         Params = []
      ),
        % This cut is essential to avoid faulty behavior (DLM):
        !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_sort_and_get_event
% purpose: Sort raw events by priority, choose the highest priority event
%    or FirstIn if equal priority, extract event data and sender,
%    and store the rest of events
% remarks:
%    The chosen event must be of HIGHER priority than LowestPriority, and
%    oaa_sort_and_get_event can fail if no appropriate event is found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_sort_and_get_event(EventList, LowestPriority, Event, Params) :-
      samsort(oaa_priority_compare, EventList, SortedList),
      oaa_choose_event(LowestPriority, SortedList, RawEvent, Remainder),
      oaa_extract_event(RawEvent, Event, Params),
      (Remainder = [] ;
       assert(oaa_event_buffer(Remainder))),
      !.

oaa_priority_compare(E1, E2) :-
      oaa_extract_event_param(E1, _, priority(P1)),
      oaa_extract_event_param(E2, _, priority(P2)),
      !, P1 >= P2.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_choose_event
% purpose: Extracts the first event from a list which has a HIGHER priority
%        than the required lowest.  Fails if none found.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_choose_event(LowestPriority, [Event|Remainder], Event, Remainder) :-
        oaa_extract_event_param(Event, _, priority(P)),
        LowestPriority < P,
        !.
oaa_choose_event(LowestPriority, [E|Rest], Event, [E|Rest2]) :-
        oaa_choose_event(LowestPriority, Rest, Event, Rest2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% name:    oaa_read_all_events
% purpose: Flush the communication event queue, reading ALL available events and
%          returning a list of them, or empty list if none available.
% remarks:
%    - Events are retrieved in raw (unextracted) form.
%    - We check to make sure the event is Validated (security hook)
%      before returning it
%    - We check to see if the event is flushed by a later event.
%      If so, we notify event sender of the flush and we don't return the
%      event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_read_all_events(TimeOut, Events, FlushPriority) :-
      oaa_select_event(TimeOut, E), !,
      (E == timeout ->
         Events = [],
         FlushPriority = 0    % lowest event priority: don't flush events
      |
         % read one event, so read all the rest
         oaa_read_all_events(0.0001, RestEvents, RestFlushPriority),

         % check if read Event is acceptable (security hook)
         (oaa_ValidateEvent(E,OkEvent) ->
            oaa_ComTraceMsg('~n[COM received]:~n   ~q~n', [OkEvent]),

            % get event's priority
            oaa_extract_event_param(OkEvent, _, priority(P)),

            % if less than some higher priority flush event, discard event
            %    and perhaps notify sender
            (P < RestFlushPriority ->
             % event will be removed,
             oaa_flush_notification(OkEvent),
             FlushPriority = RestFlushPriority,
             Events = RestEvents
            |
             % keep event: not flushed
               Events = [OkEvent|RestEvents],

             % see if this event adds a flush:
             %   if so record new flush priority
               (oaa_event_param(OkEvent, flush_events(true)) ->
                FlushPriority = P
             |  FlushPriority = RestFlushPriority)
            )

         % Not validated, skip event
         | Events = RestEvents)
      ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_ValidateEvent
% purpose: Check that an incoming lowlevel event should be processed.
%          This is the place to put security checks on events.
%    The default behavior defined by the library can be made more
%    stringent by individual agents using the callback oaa_AppValidateEvent
% remarks:
```

```
%    oaa_ValidateEvent has the right to modify the incoming event,
%    or refuse it altogether by failing.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ValidateEvent(E,OkEvent) :-
      % if oaa_AppValidateEvent is defined, use it.
      predicate_property(user:oaa_AppValidateProperty(_,_), _),
      !,
      user:oaa_AppValidateProperty(E, OkEvent).
% currently, no security checks are performed
oaa_ValidateEvent(OkEvent,OkEvent).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_flush_events
% purpose: Flushes any events with a lower priority than the FlushPriority
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_flush_events([], _FlushPriority, []).
oaa_flush_events([Event|RestEvents], FlushPriority, RemainingEvents) :-
      oaa_flush_events(RestEvents, FlushPriority, RestSaved),

      % get event's priority
      oaa_extract_event_param(Event, _, priority(P)),

      % if lower priority than we are flushing, notify and remove
      (P < FlushPriority ->
         oaa_flush_notification(Event),
         RemainingEvents = RestSaved
      |
         RemainingEvents = [Event|RestSaved]
      ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_flush_notification
% purpose: Given a raw event, grabs its real event and looks up whether
%          a notification should be sent out regarding the event's
%        cancellation due to a flush.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_flush_notification(RawEvent) :-
      oaa_extract_event(RawEvent, Event, _Params),
      (oaa_get_flush_notify(Event, NotifyEvent) ->
         oaa_PostEvent(NotifyEvent, [])
      | true), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_get_flush_notify
% purpose: Records a list of events which require a return notification
%        if the event is flushed.
% remarks:
%    currently, only the ev_solve() event returns a message;
%    all other events are flushed without notification
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @@Additional entries needed here:
oaa_get_flush_notify(ev_solve(ID, Goal, Params),
                ev_solved(ID, FromMe, Goal, Params, [])) :-
```

```
      (icl_GetParamValue(reply(none), Params) ->
         fail
      | oaa_Id(FromMe)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_select_event
% purpose: If a positive timeout is defined, wait N seconds for an event
%    to arrive
%    Otherwise block-wait until an event arrives.
% remarks: IMPORTANT: Connected/1 gets special handling, because we want
%          the connection ID and oaa ID to be assigned immediately.
%          Otherwise, oaa_translate_incoming_event and oaa_unwrap_event
%          won't always work properly for subsequent events from the
%          new connection (or would have to be more complicated).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_select_event(TimeOut, Event) :-
      com:com_SelectEvent(TimeOut, InEvent),
      ( InEvent = connected(_) ->
          oaa_ProcessEvent(InEvent, []),
          oaa_select_event(TimeOut, Event)
      | otherwise ->
          oaa_translate_incoming_event(InEvent, TranslatedEvent),
          oaa_unwrap_event(TranslatedEvent, _Connection, Event)
      ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_unwrap_event(+TranslatedEvent, -Connection, -Event).
% arguments: TranslatedEvent: An event from another agent, which has already
%            been translated for version compatibility, if necessary.
%            Event: An event term in our standard internal format, as required
%                by all other library procedures.
%            Connection: The CONNECTION of the immediate agent
%                from which this message came (note that an agent's CONNECTION
%                can be different than its ID).
% purpose: Remove an event term from its communications wrapper (if any),
%          and returns it in our standard internal form:
%          'timeout' OR event(Content, Params).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% timeout is the ONLY event that doesn't get embedded in event/2:
oaa_unwrap_event(timeout, unknown, timeout) :-
    !.
oaa_unwrap_event(term(Connection, event(Content,Params)), ConnectionId,
                 event(Content, NewParams)) :-
    !,
    ( com:com_GetInfo(ConnectionId, connection(Connection)) ->
      true
    | otherwise ->
      format(
          '~w: incoming event from an unrecognized connection (~w):~n  ~w~n',
        ['INTERNAL ERROR', Connection, event(Content,Params)]),
        ConnectionId = unknown
    ),
    ( memberchk(from(_), Params) ->
        NewParams = [connection_id(ConnectionId) | Params]
    | Content = ev_connected(InfoList),
```

```
      memberchk(fac_id(Id), InfoList) ->
        NewParams = [from(Id), connection_id(ConnectionId) | Params]
    | ConnectionId = parent,
      com:com_GetInfo(ConnectionId, fac_id(Id)) ->
        NewParams = [from(Id), connection_id(ConnectionId) | Params]
    | com:com_GetInfo(ConnectionId, oaa_id(Id)) ->
        NewParams = [from(Id), connection_id(ConnectionId) | Params]
    | otherwise ->
      % With current code, this should never happen.  But I can
      % imagine code changes that might need this (DLM 98/02/18):
        NewParams = [from(unknown), connection_id(ConnectionId) | Params]
    ).

% This handles connected/1, end_of_file/1, wakeup/1:
oaa_unwrap_event(Content, unknown, event(Content, [])).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_translate_incoming_event(+InEvent, -OutEvent).
% purpose: Provides backwards compatibility by calling a hook
%    (user:oaa_event_translation/7) that translates incoming events from agents
of
%    other versions.  Also allows for event differences based on language.
%    The idea is to return an event with both format and contents that
%    are appropriate for the agent receiving the event.
% remarks: user:oaa_event_translation/7 can be hard-coded, loaded at runtime,
%    or whatever.  If it's not present, we return the same event.
%    Note that the translation hook is somewhat limited.  It allows a single
%    event to be translated to another single event, and with essentially
%    no information about context.  This inadequate or awkward for some cases.
%    Those cases are handled using extra clauses of user:oaa_AppDoEvent (in
%    translations.pl).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Special cases.  There's no need to translate these.  And, it could be
% problematical, because we don't yet know the language and version of
% the sender.
oaa_translate_incoming_event(term(Conn, event(Contents, Params)),
                        term(Conn, event(Contents, Params))) :-
      ( Contents = ev_connect(_) ;
        Contents = ev_connected(_) ),
    !.

oaa_translate_incoming_event(term(Connection, InEvent),
                        term(Connection, OutEvent)) :-
    current_predicate(oaa_event_translation,
                  user:oaa_event_translation(_,_,_,_,_,_,_)),
    ( com:com_GetInfo(ConnectionId, connection(Connection)) ->
      true
    | otherwise ->
      true
    ),
        % These assumptions may not always be right, but will
      % nearly always get the desired results.
        % :
    ( ground(ConnectionId),
```

```
      com:com_GetInfo(ConnectionId, agent_version(PriorVersion)) ->
        true
    | otherwise ->
        PriorVersion = 2.1
    ),
    ( ground(ConnectionId),
      com:com_GetInfo(ConnectionId, agent_language(PriorLanguage)) ->
        true
    | otherwise ->
        PriorLanguage = c
    ),
    oaa_LibraryVersion(MyVersion),
    ( MyVersion \== PriorVersion ; PriorLanguage \== prolog ),
    user:oaa_event_translation(PriorVersion, PriorLanguage, MyVersion, prolog,
                          Connection, InEvent, OutEvent),
    !.
% This handles timeout/0, connected/1, end_of_file/1, wakeup/1.
% Also passes through any event for which there is no translation.
oaa_translate_incoming_event(Event, Event) :-  !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_extract_event
% purpose: Extract the content and parameters from an event term.
% remarks: Always succeeds.
%          The content part of the term is often (loosely) called the Event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_extract_event(event(Content, Params), Content, Params) :-
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_extract_event_param
% purpose:  Extract the content and a parameter value from an event term.
% remarks: Always succeeds - unless you ask for a param that has no default
%          value.
%          The content part of the term is often (loosely) called the Event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_extract_event_param(event(Content, Params), Content, Param) :- !,
        icl_GetParamValue(Param, Params).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_event_param
% purpose:  Extract a parameter from an event term.
% remarks:  This FAILS if the parameter isn't present (unlike
%           oaa_extract_event_param).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_event_param(event(_Content, Params), Param) :- !,
        memberchk(Param, Params).



%*************************************************************************
% Interpreting EVENTS
%*************************************************************************
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Interpret(+ICLExpression, +Params)
% purpose: Executes an incoming event
% remarks: Implements a simple meta-interpreter for executing complex goals.
%          Agent goals are interpreted by oaa_exec_event().
%
%          The contents of Params will vary depending on context.
%        When oaa_Interpret is called on an incoming event, Params
%        will (usually) include from(Sender).  Calls generated internally
%        may contain from(self).  Additional params may
%        accumulate through recursive calls to oaa_Interpret.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Interpret(Goal, _) :- var(Goal), !, fail.  % How could this happen?
oaa_Interpret(true, _) :- !.
oaa_Interpret(fail, _) :- !, fail.
oaa_Interpret(false, _) :- !, fail.
oaa_Interpret((\+ P), Params) :- !, \+ oaa_Interpret(P, Params).
oaa_Interpret((P -> Q ; _R), Params) :-
    oaa_Interpret(P, Params), !, oaa_Interpret(Q, Params).
oaa_Interpret((_P -> _Q ; R), Params) :- !, oaa_Interpret(R, Params).
oaa_Interpret((P -> Q), Params) :- !, oaa_Interpret((P -> Q ; fail), Params).
oaa_Interpret((X, Y), Params) :- !,
    oaa_Interpret(X, Params), oaa_Interpret(Y, Params).
oaa_Interpret((X ; Y), Params) :- !,
    (oaa_Interpret(X, Params) ; oaa_Interpret(Y, Params)).
oaa_Interpret(findall(Var, Goal, All), Params) :- !,
    findall(Var, oaa_Interpret(Goal, Params), All).
oaa_Interpret(P, _Params) :- icl_BuiltIn(P), !, call(P).
oaa_Interpret(X, Params) :- oaa_exec_event(X, Params).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_exec_event
% purpose: Defines execution of events built into all agents
% remarks: Goals that can't be handled by oaa_exec_event are passed to the
%          user-declared app_do_event callback, if present.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% turn on trace
oaa_exec_event(ev_trace_on, _) :-
      abolish(oaa_trace/1),
      assert(oaa_trace(on)),
      format('~nTrace on.~n', []), !.

% turn off trace
oaa_exec_event(ev_trace_off, _) :-
      abolish(oaa_trace/1),
      assert(oaa_trace(off)),
      format('~nTrace off.~n', []), !.

% tcp level trace
oaa_exec_event(ev_com_trace_on, _) :-
      abolish(oaa_com_trace/1),
      assert(oaa_com_trace(on)),
      format('~nCOMMUNICATION PROTOCOL trace on.~n', []), !.

% tcp level trace
```

```
oaa_exec_event(ev_com_trace_off, _) :-
      abolish(oaa_com_trace/1),
      assert(oaa_com_trace(off)),
      format('~nCOMMUNICATION PROTOCOL trace off.~n', []), !.


% turn on debug
oaa_exec_event(ev_debug_on, _) :-
      abolish(oaa_debug/1),
      assert(oaa_debug(on)),
      format('~nDebug on.~n', []), !.

% turn off debug
oaa_exec_event(ev_debug_off, _) :-
      abolish(oaa_debug/1),
      assert(oaa_debug(off)),
      format('~nDebug off.~n', []), !.

% Set the timeout value
oaa_exec_event(ev_set_timeout(N), _) :-
      abolish(timeout/1),
      assert(timeout(N)),
      format('~nTimeout set to ~q.~n', [N]), !.

% Notification that some other agent has disconnected.  Currently, this applies
% only to peer client agents, and the arg. will always be a local ID.
oaa_exec_event(ev_agent_disconnected(LID), _) :-
        oaa_remove_data_owned_by(LID).

% quit to UNIX
oaa_exec_event(ev_halt, _) :-
      format('~nDisconnecting...~n', []),
      com:com_Disconnect(parent),
      ( oaa_call_callback(app_done, _, []) ; true ),
      halt.

oaa_exec_event(ev_update(ID, Mode, Clause, Params), EvParams) :-
      oaa_Id(AgentId),
      append(Params, EvParams, AllParams),
      ( Mode = add ->
          Functor = oaa_add_data_local
      | Mode = remove ->
          Functor = oaa_remove_data_local
      | Mode = replace ->
          Functor = oaa_replace_data_local
      ),
      Call =.. [Functor, Clause, AllParams],
      ( call(Call) ->
          Updaters = [AgentId]
      | otherwise ->
          Updaters = []
      ),
      (icl_GetParamValue(reply(none), AllParams) -> true |
           oaa_PostEvent(ev_updated(ID, Mode, Clause, Params, Updaters),
                   [])
      ).
```

```
% add or remove a local trigger
oaa_exec_event(ev_update_trigger(ID, Mode, Type,
                        Condition, Action, TrigParams),
             Params) :-
      oaa_Id(AgentId),
      append(TrigParams, Params, NewParams),
      ( Mode == add ->
          Functor = oaa_add_trigger_local
      | Mode == remove ->
          Functor = oaa_remove_trigger_local
      ),
      Call =.. [Functor, Type, Condition, Action, NewParams],
      ( call(Call) ->
          Updaters = [AgentId]
      | otherwise ->
          Updaters = []
      ),
      ( icl_GetParamValue(reply(none), Params) ->
         true
      | otherwise ->
          oaa_PostEvent(ev_trigger_updated(ID, Mode, Type, Condition,
                                  Action, TrigParams, Updaters),
                  [])
      ),
      ( Mode = add ->
           oaa_Inform(trigger, 'trigger_added(~q,~q,~q,~q)~n',
                  [Type, Condition, Action, NewParams])
      | true
      ).

% When asked to solve a goal, see if you know how to solve
% it, then find all solutions.  Send the solutions to the
% caller.
%
% The various params lists must be used with care.  Searching different
% lists may be appropriate for different params, depending on their
% meanings.  Another consideration is that Solve params and Goal params,
% as returned to the requesting agent, must unify with the original
% lists that came from the requesting agent.

oaa_exec_event(ev_solve(ID, FullGoal, SolveParams), Params) :-
      oaa_class(leaf),
        icl_GoalComponents(FullGoal, _, _, GoalParams),

        % More "local" params take precedence, so they go to the
      % beginning of the list:
      append([SolveParams, Params], InheritedParams),
      append([GoalParams, InheritedParams], AllParams),
      % Assert context:
      findall(context(C), member(context(C), AllParams), Contexts),
      asserta( oaa_current_contexts(ID, Contexts) ),

      oaa_TraceMsg('~n~nAttempting to solve:~n  Goal:~q~n  Params:~q~n',
                  [FullGoal, InheritedParams]),
        findall(FullGoal,
                oaa_solve_local(FullGoal, InheritedParams),
            Solutions),
```

```
        oaa_TraceMsg('~nSolutions found for ~q:~n   ~q~n',
                     [FullGoal, Solutions]),

      % If user has requested to delay the solution (oaaDelaySolution)
      % save current userId, Goal and Params in delay table, to be
      % sent back in an ev_solved() msg later (oaaReturnDelayedSolutions).

      (retract(oaa_delay(ID, UserId)) ->
         assert(oaa_delay_table(ID, UserId, FullGoal, SolveParams, AllParams))
      |
         (icl_GetParamValue(reply(none), AllParams) -> true |
            (oaa_Id(FromKS) ; FromKS = unknown), !,
              oaa_PostEvent(ev_solved(ID, FromKS, FullGoal, SolveParams,
                                      Solutions),[])
         )
      ),

      % Retract context:
      retractall( oaa_current_contexts(ID, _) ).


% This is for subgoals (of goals passed in solve events) that have
% Params.  Subgoals with no params will fall through to the next clause.
oaa_exec_event(Goal::GoalParams, Params) :-
        oaa_solve_local(Goal::GoalParams, Params).

% call user events.  Must not have a cut, to return all solutions.
oaa_exec_event(Event, Params) :-
      oaa_turn_on_debug,
      ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
      ( (oaa_goal_matches_solvables(Event, Solvables, Goal, Matched),
         Matched = solvable(_, SolvParams, _),
         (icl_GetParamValue(callback(CB), SolvParams) ;
          oaa_callback(app_do_event, CB)))
      ;
        (oaa_callback(app_do_event, CB),
         Goal = Event)
      ),
      !,
      ( CB = Module:Functor ->
          true
      | otherwise ->
          Module = user,
          Functor = CB
      ),
      Call =.. [Functor, Goal, Params],
      on_exception(E,
         Module:Call,
         ( oaa_TraceMsg('WARNING (agent.pl): Exception raised thru callback
handler (~w):~n  ~q~n',
                        [Functor, E]),
           fail )),
      oaa_turn_off_debug.


% What to do about test(TEST)?
% if test(TEST) is listed in arguments, solve
```

```
%    it locally.
passes_tests(Params) :-
      oaa_class(leaf),
      icl_GetParamValue(test(Test), Params),
      !,
      oaa_Solve(Test, [level_limit(0)]).
% With compound goals, we also want to allow tests on the facilitator.
% @@DLM: Is this the best way?
passes_tests(Params) :-
      (oaa_class(root);oaa_class(node)),
      icl_GetParamValue(test(Test), Params),
      !,
      oaa_solve_local(Test, []).
passes_tests(_Params) :-
      true.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_DelaySolution
% purpose: Requests that the current AppDoEvent not return solutions to the
%        current goal until a later time.
%  inputs:
%   - Id: an Id which will be used to later match solutions to request
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_DelaySolution(Id) :-
      oaa_current_contexts(GoalId, _Contexts), !,
      assert(oaa_delay(GoalId, Id)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_ReturnDelayedSolutions
% purpose: Returns the list of solutions for a delayed request
%  inputs:
%   - Id: an Id referring to a previously saved oaa_DelaySolution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ReturnDelayedSolutions(Id, SolutionList) :-
      (retract(oaa_delay_table(GoalId, Id, Goal, SolveParams,AllParams)) ->
         (icl_GetParamValue(reply(none), AllParams) -> true |
            (oaa_Id(FromKS) ; FromKS = unknown), !,
            % make sure all Solutions unify with original goal
            findall(Goal, member(Goal,SolutionList), Solutions),
              oaa_PostEvent(ev_solved(GoalId, FromKS, Goal, SolveParams,
                                Solutions),[])
         )
      | true).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_AddDelayedContextParams
% purpose: When a goal is delayed using oaa_DelaySolution(), incoming context
%        parameters from the original request can not be automatically
%        concatenated to outgoing oaa_Solve requests -- since an agent can
%        manage multiple delayed goals at the same time, liboaa doesn't
%        know the correct context for the outgoing oaa_Solve without explicit
%        direction from the programmer.  Hence, an agent programmer who
%        wants to call oaa_Solve during a delayed goal is expected to
%        use this function to add the saved contexts for the delayed goal to
```

```
%        his/her outgoing oaa_Solve parameters.
%  inputs:
%   - Id: an Id which will be used to later match solutions to request
%   - Params: Parameters for solve goal
%   - NewParams: Params augmented by saved contexts.
%  example:
%      oaa_AppDoEvent(goal(_X),_Params) :-  oaa_DelayEvent(a_goal).
%      oaa_AppDoEvent(temp_event(Y),_Params) :-
%           oaa_AddDelayedContextParams(a_goal, [], P),
%           oaa_Solve(sub_goal(Y), P).
%      oaa_AppDoEvent(final_event(S), _Params) :-
%           oaa_ReturnDelayedSolutions(a_goal, [goal(S)]).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddDelayedContextParams(Id, Params, NewParams) :-
      retract(oaa_delay_table(_GoalId, Id, _Goal, _SolveParams, AllParams)),
      findall(context(C), member(context(C), AllParams), Contexts),
      append(Contexts, Params, NewParams).



%*****************************************************************************
% Agent-Facilitator communication
%*****************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_PostEvent
% purpose: Sends a low-level event to another agent
% remarks:
%    Should NOT be used before there's a connection established for
%      the destination (such as when a client sends ev_connect to its
%      facilitator).  In such unusual cases, use com_SendData directly.
%      For application developers, this just means don't call
%      oaa_PostEvent until after you've called oaa_Register.
%    Parameters may include:
%      - priority(P):
%      - address(A): specify address of specific server or client agent
%        A must be an agent ID, not a name.  If caller is a client agent,
%        the only meaningful address is that of the client's facilitator.
%      - from(KS): where the event originally originated
%    IMPORTANT: there may be a different address INSIDE the event;
%      these should not be confused!
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_PostEvent(Contents, Params) :-

      % see if any params of interest
        (memberchk(priority(_P), Params);
       memberchk(from(_Agent), Params) ->
          SendEvent = event(Contents, Params)
      |
          SendEvent = event(Contents, [])
      ),

      % find destination: if none, dest = server
      (memberchk(address(Dest), Params) ->
         true
```

```
      |
         Dest = parent
      ),

      icl_true_id(Dest, DestId),
        oaa_translate_outgoing_event(SendEvent, DestId, TransEvent),

      oaa_ComTraceMsg('~n[COM send to ~q]:~n   ~q~n', [Dest, TransEvent]),

      oaa_convert_id_to_comm_id(DestId, CommId),
      % send event to destination
      com:com_SendData(CommId, TransEvent),

      % Use SendEvent here, becuase triggers always contain event/2
      % to unify with.
        oaa_CheckTriggers(comm, SendEvent, send).


oaa_convert_id_to_comm_id(Id, CId) :-
      com:com_GetInfo(CId, fac_id(Id)), !.
oaa_convert_id_to_comm_id(Id, CId) :-
      com:com_GetInfo(CId, oaa_id(Id)), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_translate_outgoing_event(+Event, +DestId, -NewEvent).
% purpose: Provides backwards compatibility by calling a hook
%    (user:oaa_event_translation/7) that translates outgoing events to agents of
%    other versions.  Also allows for event differences based on language.
% remarks: user:oaa_event_translation/7 can be hard-coded, loaded at runtime,
%    or whatever.  If it's not present, we return the same event.
%    See also comments for oaa_translate_incoming_event.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Special cases.  There's no need to translate these.  And, it could be
% problematical, because we don't yet know the language and version of
% the receiver.  See comments for oaa_unwrap_event.
oaa_translate_outgoing_event(event(Contents, Params), _DestId,
                        event(Contents, Params)) :-
      ( Contents = ev_connect(_) ;
        Contents = ev_connected(_) ),
    !.
oaa_translate_outgoing_event(event(Content, Params), DestId, TransEvent) :-
    current_predicate(oaa_event_translation,
                  user:oaa_event_translation(_,_,_,_,_,_,_)),
        % These assumptions may not always be right, but will
      % nearly always get the desired results:
    com:com_GetInfo(Connection, oaa_id(DestId)),
    ( com:com_GetInfo(Connection, agent_version(DestVersion)) ->
        true
    | otherwise ->
        DestVersion = 2.1
    ),
    ( com:com_GetInfo(Connection, agent_language(DestLanguage)) ->
        true
    | otherwise ->
        DestLanguage = c
```

```
    ),
    oaa_LibraryVersion(MyVersion),
    user:oaa_event_translation(MyVersion, prolog, DestVersion, DestLanguage,
                          Connection, event(Content, Params), TransEvent),
    !.
oaa_translate_outgoing_event(Event, _, Event).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Version
% purpose: Lookup the language and library version number for an agent
% remarks: The default version (if unspecified) is 1.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_Version(AgentId, Language, Version) :-
   icl_true_id(AgentId, TrueId),
   % Asking for my version:
   oaa_Id(TrueId),
   Language = prolog,
   oaa_LibraryVersion(Version),
   !.
oaa_Version(AgentId, Language, Version) :-
   icl_true_id(AgentId, TrueId),
   ( com:com_GetInfo(CommId, oaa_id(TrueId)) ;
     com:com_GetInfo(CommId, fac_id(TrueId)) ),
   ( com:com_GetInfo(CommId, agent_language(Language)) ->
      true
   | otherwise ->
      Language = unknown
   ),
   ( com:com_GetInfo(CommId, agent_version(Version)) ->
      true
   | otherwise ->
      Version = 1.0
   ),
   !.
oaa_Version(AgentId, Language, Version) :-
    (oaa_class(leaf) ; oaa_class(node)),
    icl_true_id(AgentId, TrueId),
      % The use of caching here could be dangerous - unless we install a
      % mechanism for automatic updating of the cache.
    oaa_Solve(agent_version(TrueId, Language, Version),
              [address(parent)]),
   !.
oaa_Version(_, prolog, 1.0).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_CanSolve
% purpose: Asks the Facilitator for a list of agents which could solve a Goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_CanSolve(Goal,KSList) :-
    oaa_Solve(can_solve(Goal, KSList), [address(parent)]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Ping
```

```
% purpose: Tests whether a given agent is currently responding to requests.
% inputs:
%   AgentAddr: address of agent to test
%   TimeLimit: Time limit (in seconds) for how long to wait for a response
% outputs:
%   TotalResponseTime for round trip (in seconds)
% remarks: Fails if a ping is not returned in TimeLimit amount of time
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Ping(AgentAddr, TimeLimit, TotalResponseTime) :-
    ground(AgentAddr),
    number(TimeLimit),
    TimeLimit >= 0,
    tcp_now(Before),
    oaa_Solve(true, [address(AgentAddr), time_limit(TimeLimit)]),
    tcp_now(After),
    tcp_time_plus(Before, TotalResponseTimeMs, After),
    TotalResponseTime is TotalResponseTimeMs / 1000.



%**************************************************************************
% Declaring Solvables
%**************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Declare(+Solvables, +CommonPermissions, +CommonParams, +Params,
%                      -DeclaredSolvables)
% purpose: Declare solvables for a client or facilitator, and inform the
%          parent if appropriate.
% arguments:
%   Solvables: A single solvable or a list of solvables, in shorthand or
%          standard form.
%   CommonPermissions: Permissions to be distributed to each solvable in
%          Solvables.  This is purely for programming convenience. See
%          comments for icl_ConvertSolvables for possible values, and
%          solvables documentation for their meanings.
%   CommonParams: Params to be distributed to each solvable in Solvables.
%          This is purely for programming convenience. See comments for
%          icl_ConvertSolvables for possible values, and solvables
%          documentation for their meanings.
%   Params:
%      address(X): Where the solvable will exist.  X may be either 'self'
%          or 'parent' (or the appropriate local ids).  Default: 'self'.
%      if_exists(OverwriteOrAppend): What to do when declaring solvables
%          for self, and some already exist. Default: append.
%   DeclaredSolvables: Returns a list, in standard form, of all solvables
%      successfully declared.
% remarks:
%    - Any agent can declare solvables for itself. In addition, a client can
%      ask its facilitator to declare solvables.  Client-requested facilitator
%      solvables will automatically acquire permission write(true), and params
%      type(data), rules_ok(false), private(false), and bookkeeping(true).
%    - If called by a leaf or node agent, assumes agent is already registered
%      with a parent facilitator.
%    - Predicates can only be declared once.  Changing an existing
%      predicate definition should be done with oaa_Redeclare.  However,
```

```
%       a request to declare a predicate, which is already declared in
%       precisely the same way, succeeds transparently.
%     - @@Future params may include 'num_context_args(N)'.
%     - @@Future solvable params may include 'shared'.
%     - synonym predicates can have their own triggers, but share the clause
%       database with their master table.
%     - views and filters, as provided by the OAA V1 DB agent, are not
%       supported as separate params, but the same functionality is available
%       using other params.
%     - @@Do we want client agents to request declarations on other client
%       agents?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Declare(Solvable, InitialCommonPerms, InitialCommonParams,
          InitialParams, DeclaredSolvables) :-
    ( is_list(Solvable) ->
      SolvableList = Solvable
    | otherwise ->
      SolvableList = [Solvable]
    ),
    icl_ConvertSolvables(SolvableList, Solvables),
    icl_standardize_perms(InitialCommonPerms, false, CommonPerms),
    icl_standardize_params(InitialCommonParams, false, CommonParams),
    icl_standardize_params(InitialParams, false, Params),
    oaa_distribute_perms(Solvables, CommonPerms, Solvables1),
    oaa_distribute_params(Solvables1, CommonParams, NewSolvables),
    oaa_declare_aux(add, NewSolvables, Params, DeclaredSolvables).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_DeclareData(+Solvables, +Params, -DeclaredSolvables)
% purpose: Declare data solvables for an agent.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_DeclareData(Solv, Params, DeclaredSolvs) :-
    \+ is_list(Solv),
    !,
    oaa_DeclareData([Solv], Params, DeclaredSolvs).
oaa_DeclareData(Solvs, Params, DeclaredSolvs) :-
    % It's only necessary to specify the non-default perms and params.
    CommonPerms = [write(true)],
    CommonParams = [type(data)],
    oaa_Declare(Solvs, CommonPerms, CommonParams, Params, DeclaredSolvs).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Undeclare(+Solvables, +Params, -UndeclaredSolvables)
% purpose: Remove solvables from a client or facilitator, and inform the
%          parent if appropriate.
% arguments:
%   Solvables: A single solvable or a list of solvables, in shorthand or
%      standard form.  If a solvable is in standard form, however, ONLY
%       the goal is considered in selecting the solvables to be removed
%      (permissions and parameters are ignored).
%   Params:
%      address(X): Where the solvable exists.  X may be either 'self'
%          or 'parent' (or the appropriate local ids).  Default: 'self'.
%   DeclaredSolvables: Returns a list, in standard form, of all solvables
%      successfully removed.
```

```
% remarks:
%     - If called by a leaf or node agent, assumes agent is already registered
%       with a parent facilitator.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Undeclare(Solvable, InitialParams, UndeclaredSolvables) :-
    ( is_list(Solvable) ->
      SolvableList = Solvable
    | otherwise ->
      SolvableList = [Solvable]
    ),
    icl_minimally_instantiate_solvables(SolvableList, Solvables),
    icl_standardize_params(InitialParams, false, Params),
    oaa_declare_aux(remove, Solvables, Params, UndeclaredSolvables).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Redeclare(+Solvable, +NewSolvable, +Params)
% purpose: Replace a solvable on a client or facilitator, and inform the
%          parent if appropriate.
% arguments:
%   Solvable: A single solvable, in shorthand or standard form.  If in
%      standard form, however, ONLY the goal is considered in selecting
%      the solvable to be replaced (permissions and parameters are ignored).
%   NewSolvable: A single solvable, in shorthand or standard form.
%   Params:
%      address(X): Where the solvable exists.  X may be either 'self'
%          or 'parent' (or the appropriate local ids).  Default: 'self'.
% remarks:
%     - If called by a leaf or node agent, assumes agent is already registered
%       with a parent facilitator.
%     - FAILS if the operation cannot be completed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Redeclare(InitialSolvable, InitialNewSolvable, InitialParams) :-
    icl_minimally_instantiate_solvables([InitialSolvable], [Solvable]),
    icl_ConvertSolvables([InitialNewSolvable], [NewSolvable]),
    icl_standardize_params(InitialParams, false, Params),
    oaa_declare_aux(replace, Solvable, [with(NewSolvable) | Params],
                RedeclaredSolvables),
    RedeclaredSolvables \== [].


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_declare_aux(+Mode, +Solvables, +Params, -DeclaredSolvables)
% purpose: Common code for oaa_Declare, oaa_Undeclare, oaa_Redeclare.
% Mode: add, remove, or replace.
% Solvables: for Mode = add, a list of Solvables in standard form.
%            for Mode = remove, a list of Solvables in "minimally instantiated"
%                form.
%            for Mode = replace, a list containing a single Solvable, in
%                "minimally instantiated" form.
% Params: whatever is appropriate for oaa_Declare, _Undeclare, _Redeclare.
%           Must already be in standard form.
% DeclaredSolvables: A list of all solvables successfully added (or removed
%           or replaced), in standard form.
% remarks:
%   A number of params and perms are required when requesting that a
%   parent declare solvables (see comments for oaa_Declare).  We could ensure
```

```
%   their presence here, but it's not essential, because the facilitator will
%   enforce this.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here, a client is asking the facilitator to add, remove, or replace
% solvables.
oaa_declare_aux(Mode, Solvables, Params, DeclaredSolvables) :-
    com:com_GetInfo(parent, fac_id(ParentId)),
    memberchk(address([ParentId]), Params),
    !,
    % Send the request to the Facilitator
    oaa_PostEvent(ev_post_declare(Mode, Solvables, Params), []),
    oaa_poll_until_event(
        ev_reply_declared(Mode, Solvables, Params, DeclaredSolvables)).

% Leaf, node or root adding, removing or replacing its own solvables:
oaa_declare_aux(Mode, Solvables, Params, DeclaredSolvables) :-
    oaa_Id(Me),
    ( memberchk(address(Addr), Params) ->
        Addr = [Me]
    | true),
    !,

    oaa_declare_local(Mode, Solvables, Params, DeclaredSolvables),

    % If I'm a facilitator, I must also "register" my Solvables with myself.
    % (If I'm a node, this will also register them with my parent.)
    ( (\+ oaa_class(leaf), DeclaredSolvables \== []) ->
        oaa_Name(MyName),
          user:oaa_AppDoEvent(
            ev_register_solvables(Mode, DeclaredSolvables, MyName, Params),
          [from(Me)])
    | true
    ),

    % If I'm a leaf, post public solvables to parent facilitator:
    select_elements(DeclaredSolvables, oaa_public_solvable, PublicSolvables),
    ( (oaa_class(leaf), PublicSolvables \== []) ->
      com:com_GetInfo(parent, oaa_name(MyNameC)),
        oaa_PostEvent(
          ev_register_solvables(Mode, PublicSolvables, MyNameC, Params),
          [])
    | true ).

    % Solvable must be in standard form.
oaa_public_solvable(solvable(_Solvable, Params, _Perms)) :-
    icl_GetParamValue(private(false), Params).

    % Solvable must be in standard form.
oaa_data_solvable(solvable(_Solvable, Params, _Perms)) :-
    icl_GetParamValue(type(data), Params).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_declare_local(+Mode, +Solvables, +Params, -DeclaredSolvables)
% purpose:   Declare solvables for an agent.
% Mode:      add, remove, or replace.
% Solvables: The form they're in depends on the mode.  See oaa_declare_aux.
```

```
% DeclaredSolvables: Returns those members of Solvables for which
%     the operation was successful (more specifically, those that should
%     be passed up to the parent in ev_register_solvables).  Always returned
%     in STANDARD FORM.
% Also see:  comments for oaa_Declare, oaa_Undeclare, oaa_Redeclare.
% remarks:
%   - This performs the local processing needed by calls to oaa_Declare,
%     and by ev_declare events.
%   - Solvables and Params must already be in standard form.
%
%   @@DLM: Could do more careful testing to be sure the solvables are
%   all valid for the requested operation.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_declare_local(Mode, Solvable, Params, DeclaredSolvables) :-
    \+ is_list(Solvable),
    !,
    oaa_declare_local(Mode, [Solvable], Params, DeclaredSolvables).
oaa_declare_local(add, InitialSolvables, Params, DeclaredSolvables) :-
    ( icl_GetParamValue(if_exists(overwrite), Params) ->
      CurrentSolvables = []
    | oaa_solvables(CurrentSolvables) ->
      true
    | CurrentSolvables = []
    ),
    % This will eliminate those that unify with an already declared solvable.
    % @@DLM: Should do more, though: warnings.
    solvables_to_be_added(InitialSolvables, CurrentSolvables,
                                DeclaredSolvables),

    % Make sure Quintus has the correct properties for each DB solvable.
    select_elements(DeclaredSolvables, oaa_data_solvable, DBSolvables),
    oaa_declare_for_prolog(DBSolvables),

    append(CurrentSolvables, DeclaredSolvables, AllSolvables),
    retractall(oaa_solvables(_)),
    assert(oaa_solvables(AllSolvables)).

oaa_declare_local(remove, Solvables, _Params, RemovedSolvables) :-
    % See which ones are really declared:
    ( oaa_solvables(Current) -> true | Current = [] ),
    solvables_to_be_removed(Solvables, Current, RemovedSolvables),
    % Retract all clauses from data solvables:
    select_elements(RemovedSolvables, oaa_data_solvable, DBSolvables),
    oaa_remove_solvables_data(DBSolvables),
    % Assert the new solvables list:
    retractall(oaa_solvables(_)),
    subtract(Current, RemovedSolvables, New),
    assert(oaa_solvables(New)).

oaa_declare_local(replace, [Solvable], Params, [Solvable]) :-
    memberchk(with(NewSolvable), Params),
    % Make sure Solvable is really declared:
    ( oaa_solvables(Current) -> true | otherwise -> Current = []),
    memberchk(Solvable, Current),
    !,
    % If a data solvable, maybe retract all its clauses:
    ( oaa_data_solvable(Solvable) ->
```

```
        oaa_remove_solvables_data([Solvable])
    | true
    ),
    % Assert the new solvables list:
    retractall(oaa_solvables(_)),
    replace_element(Solvable, Current, NewSolvable, New),
    assert(oaa_solvables(New)).
oaa_declare_local(replace, [Solvable], _Params, []) :-
    Solvable = solvable(Goal, _, _),
    format('~w: Ignoring attempt to replace a non-existent solvable:~n  ~w~n',
         ['WARNING', Goal]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_distribute_params(+Solvables, +CommonParams, -NewSolvables).
%       oaa_distribute_perms(+Solvables, +CommonPerms, -NewSolvables).
% purpose: Add CommonParams (CommonPerms) to the Params (Permissions) list of
%       each solvable in Solvables.
% Solvables: a solvables list, in standard form.
% remarks: @@Should warn when a solvables has a param that conflicts with
%          CommonParams.  Also, should have an arg that says which version of
%          of the conflicting param to keep.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_distribute_params([], _CommonParams, []).
oaa_distribute_params([Solvable | Solvables], CommonParams,
                  [NewSolvable | NewSolvables]) :-
    Solvable = solvable(Goal, Params, Perms),
    union(Params, CommonParams, NewParams),
    NewSolvable = solvable(Goal, NewParams, Perms),
    oaa_distribute_params(Solvables, CommonParams, NewSolvables).

oaa_distribute_perms([], _CommonPerms, []).
oaa_distribute_perms([Solvable | Solvables], CommonPerms,
                  [NewSolvable | NewSolvables]) :-
    Solvable = solvable(Goal, Params, Perms),
    union(Perms, CommonPerms, NewPerms),
    NewSolvable = solvable(Goal, Params, NewPerms),
    oaa_distribute_perms(Solvables, CommonPerms, NewSolvables).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: solvables_to_be_added(+ProposedSolvs, +CurrentSolvs, -SolvsToBeAdded).
% purpose: Checks a list of solvables, to make sure they can legally be
%          declared.
% ProposedSolvs: Must be in STANDARD FORM.
% CurrentSolvs: This agent's current solvables.
% SolvsToBeAdded: A subset of ProposedSolvs.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solvables_to_be_added([], _Current, []).
solvables_to_be_added([Solvable | Solvables], Current, OKSolvables) :-
    Solvable = solvable(Goal, _, _),
    memberchk(solvable(Goal, _, _), Current),
    !,
    format('~w: Ignoring attempt to declare an already existing solvable:~n
~w~n',
         ['WARNING', Goal]),
    solvables_to_be_added(Solvables, Current, OKSolvables).
```

```
solvables_to_be_added([Solvable | Solvables], Current,
                      [Solvable | OKSolvables]) :-
    solvables_to_be_added(Solvables, Current, OKSolvables).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: solvables_to_be_removed(+ProposedSolvs, +CurrentSolvs,
%                               -SolvsToBeRemoved).
% purpose: Checks a list of solvables, to make sure they can legally be
%          UNdeclared.
% ProposedSolvs: Must be in MINIMALLY INSTANTIATED FORM.
% CurrentSolvs: This agent's current solvables.
% SolvsToBeRemoved: A subset of ProposedSolvs, but returned in standard form,
%   fully instantiated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solvables_to_be_removed([], _Current, []).
solvables_to_be_removed([Solvable | Solvables], Current,
                  [Solvable | OKSolvables]) :-
    memberchk(Solvable, Current),
    !,
    solvables_to_be_removed(Solvables, Current, OKSolvables).
solvables_to_be_removed([Solvable | Solvables], Current, OKSolvables) :-
    Solvable = solvable(Goal, _, _),
    format('~w: Ignoring attempt to remove a non-existent solvable:~n  ~w~n',
         ['WARNING', Goal]),
    solvables_to_be_removed(Solvables, Current, OKSolvables).



%*****************************************************************************
% Updating Data Solvables
%*****************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_AddData(+Clause, +Params).
% purpose: Add a new clause for a DATA solvable (locally and/or remotely)
% Params:
%    address(X): a list including 'self', 'parent', and/or the
%      addresses of other client agents.  The default (no address)
%      behavior is the same as with oaa_Solve.
%    reflexive(T_F): Save as with oaa_Solve.  Default: true.
%    at_beginning(T_F): if true, uses asserta instead of assertz.
%      Default: false.
%    single_value(T_F): if true, ALL clauses for this predicate are removed
%      before adding the new clause.
%      Default: false.
%    unique_values(T_F): if true, at most one copy of each value is stored.
%      Default: false.
%    owner(LocalId): if bookkeeping(true) for this solvable, record
%      LocalId as the owner.
%      Default: the agent from which the request originated.
%    get_address(X): Returns a list of addresses (ids) of agents that
%      were sent the request.
%    get_satisfiers(X): Returns a list of addresses (ids) of agents that
%      successfully completed the request.
```

```
%    reply({true,none}): When data is being added on
%      a remote agent or agents, this tells whether reply message(s) are
%      desired.
%     block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block.  In
%                         this case, the reply events (ev_reply_updated)
%                         can be handled by the user's app_do_event callback
%                 Default: true.  Note that reply(none) overrides
%                         block(true).
% remarks:
%    - Clause is normally a fact (no body), but with Prolog agents, and
%      with rules_ok(true), it's possible for it to have a body.
%    - Triggers will be examined with the on(add) operation mask
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddData(Clause, Params) :-
    oaa_update(add, Clause, Params).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_RemoveData(+Clause, +Params).
% purpose: Remove a clause from a DATA solvable (locally and/or remotely)
% Params:
%    address(X): a list including 'self', 'parent', and/or the
%      addresses of other client agents.  The default (no address)
%      behavior is the same as with oaa_Solve and oaa_AddData.
%    reflexive(T_F): Save as with oaa_Solve.  Default: true.
%    do_all(T_F): If true, removes all predicate values that match the Clause
%      Default: false (removes only the first)
%    get_address(X): Returns a list of addresses (ids) of agents that
%      were sent the request.
%    get_satisfiers(X): Returns a list of addresses (ids) of agents that
%      successfully completed the request.
%    owner(LocalId): if bookkeeping(true) for this solvable, remove only
%      data owned by LocalId.
%      Default: ignore owner in removing data.
%    reply({true,none}): When data is being removed on
%      a remote agent or agents, this tells whether reply message(s) are
%      desired.
%     block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block.  In
%                         this case, the reply events (ev_reply_updated)
%                         can be handled by the user's app_do_event callback
%                 Default: true.  Note that reply(none) overrides
%                         block(true).
% remarks:
%    - Clause is normally a fact (no body), but with Prolog agents, and
%      with rules_ok(true), it's possible for it to have a body.
%    - Triggers will be examined with the 'on_Retract' operation mask.
%    - Not for backtracking.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_RemoveData(Clause, Params) :-
    oaa_update(remove, Clause, Params).


%-------------------------------------------------------------------------
% name:    oaa_ReplaceData(+Clause1, +Clause2, +Params).
% purpose: Change a predicate value to a new one
```

```
% Clause1: Must be a clause of a writable data solvable.
% Clause2: Must be a clause of a writable data solvable.
% Params:
%    address(X): a list including 'self', 'parent', and/or the
%      addresses of other client agents.  The default (no address)
%      behavior is the same as with oaa_Solve and oaa_AddData.
%    reflexive(T_F): Save as with oaa_Solve.  Default: true.
%    do_all(T_F): If, true, changes all predicate values that match the
%            Clause1 specification
%            default is 'false': changes only the first
%    at_beginning(T_F): If true, uses asserta instead of assertz
%            default is 'false'
%    owner(LocalId): if bookkeeping(true) for this solvable, record
%      LocalId as the owner of each new data item.  Note: It is not possible
%      to specify the owner of the data to be replaced, just that of the
%      NEW data.
%      Default: the agent from which the request originated.
%    get_address(X): Returns a list of addresses (ids) of agents that
%      were sent the request.
%    get_satisfiers(X): Returns a list of addresses (ids) of agents that
%      successfully completed the request.
%    reply({true,none}): When data is being replaced on
%      a remote agent or agents, this tells whether reply message(s) are
%      desired.
%    block(Mode)   : true: Block until the reply arrives.
%                  : false: Don't block.  In
%                          this case, the reply events (ev_reply_updated)
%                          can be handled by the user's app_do_event callback
%                 Default: true.  Note that reply(none) overrides
%                          block(true).
% remarks:
%    - Clause1 and/or Clause2 may be synonym predicates.
%    - Clause1 and Clause2 are not required to have the same functor.
%    - Clause1 and Clause2 may share variables.
%    - Triggers will be examined with the 'remove' operation mask with Clause1,
%      and the 'add' operation mask with Clause2.
%    - db_replace triggers on the Pred2 argument, not on the Pred1 arg
%    - at_beginning param only used if do_all is false
%-------------------------------------------------------------------------
oaa_ReplaceData(Clause1, Clause2, Params) :-
    oaa_update(replace, Clause1, [with(Clause2) | Params]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:     oaa_update(+Mode, +Clause, +Params).
% purpose:  Common code for oaa_AddData, oaaRemoveData, and oaa_ReplaceData.
% Mode:     add, remove, or replace.
% Clause, Params: May include whatever is appropriate for oaa_AddData,
%                 oaaRemoveData, or oaa_ReplaceData.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_update(Mode, Clause, InitialParams) :-
   icl_standardize_params(InitialParams, false, Params),
   % Is there a specified address?
   ( memberchk(address(Addr), Params) ->
       true
   | otherwise ->
       Addr = []
```

```
),

% Decide whether or not to update locally:
oaa_Id(Me),
( memberchk(Me, Addr) ->
    delete(Addr, Me, NewAddr),
    replace_element(address(Addr), Params, address(NewAddr), Params1),
    Self = true
| otherwise ->
    NewAddr = Addr,
    Params1 = Params
),
( Addr = [], icl_GetParamValue(reflexive(true), Params1) ->
    % do NOT use remove_element here:
    delete(Params1, reflexive(true), Params2),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = [] ),
    ( oaa_data_matches_solvables(Clause, Solvables, write, _, _) ->
        Self = true
    | otherwise ->
        true
    )
| otherwise ->
    Params2 = Params1
),

% Update locally if appropriate:
( Self == true ->
    Requestees1 = [Me],
    ( Mode == add ->
        Functor = oaa_add_data_local
    | Mode == replace ->
        Functor = oaa_replace_data_local
    | Mode == remove ->
        Functor = oaa_remove_data_local
    ),
    LocalCall =.. [Functor, Clause, Params2],
    ( call(LocalCall) ->
        Updaters1 = [Me]
    |   Updaters1 = [])
| otherwise ->
    Requestees1 = [],
    Updaters1 = []
),

% Update remotely if appropriate:
( oaa_class(leaf), (Addr == [] ; NewAddr \== []) ->
    % Send the ev_post_update event to the Facilitator
    oaa_PostEvent(ev_post_update(Mode, Clause, Params2), []),
    % In the return event, Requestee2s lists all agents to whom
    % the update request was sent; Updaters2 lists those who succeeded.
    ( (icl_GetParamValue(reply(asynchronous), Params) ;
       icl_GetParamValue(reply(none), Params)) ->
        Requestees2 = [],
        Updaters2 = []
    | otherwise ->
        oaa_poll_until_event(
          ev_reply_updated(Mode, Clause, Params2, Requestees2, Updaters2))
```

```
    )
  | otherwise ->
    Requestees2 = [],
    Updaters2 = []
  ),
  append(Updaters1, Updaters2, Updaters),
  % Return Updaters if requested:
  ( memberchk(get_satisfiers(Updaters), Params) -> true | true ),
  append(Requestees1, Requestees2, Requestees),
  % Return Requestees if requested:
  ( memberchk(get_address(Requestees), Params) -> true | true ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:      oaa_add_data_local(+Clause, +Params)
% purpose:   Assert a clause for an agent's solvable.
% arguments: See comments for oaa_AddData.
% remarks:
%   This performs the local processing needed for calls to oaa_AddData, and
%   ev_update(add, ...) requests.
%   Application code should not call oaa_add_data_local directly, but rather
%   oaa_AddData with address(self).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_add_data_local(Clause1, Params) :-
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1, Solvables, write, Clause, Matched),
    Matched = solvable(Pred, DeclParams, _Perms),
    ( Clause = (Head :- Body) ->
        true
    | otherwise ->
        Head = Clause,
        Body = true
    ),

    append(Params, DeclParams, AllParams),
    % If there's no callback, leave Callback a var:
    ( memberchk(callback(Callback), AllParams) -> true | true ),

    % if single value, erase all old values
    (icl_GetParamValue(single_value(true), AllParams) ->
        ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
          oaa_retractall((Pred :- _), _OldOwner, Callback)
      | otherwise ->
          retract_all((Pred :- _))
      )
    | true),

    % if unique_values(true), make sure fact not already in database
    ( clause(Head, Body), icl_GetParamValue(unique_values(true), AllParams) ->
        true
    | otherwise ->
        ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
          oaa_data_owner(Params, Owner),
            ( icl_GetParamValue(at_beginning(true), AllParams) ->
                oaa_asserta(Clause, Owner, Callback)
            |
                oaa_assertz(Clause, Owner, Callback)
```

```
            )
        | otherwise ->
            ( icl_GetParamValue(at_beginning(true), AllParams) ->
                asserta(Clause)
            |
                assertz(Clause)
            )
        )
    ),
    oaa_CheckTriggers(data, Head, add),
   !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:       oaa_remove_data_local(+Clause, +Params)
% purpose:    Retract a clause (or all clauses) from an agent's solvable.
% arguments: See comments for oaaRemoveData.
% remarks:
%    This performs the local processing needed for calls to oaaRemoveData, and
%    ev_update(remove, ...) requests.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_remove_data_local(Clause1, Params) :-
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1, Solvables, write, Clause, Matched),
    Matched = solvable(_Pred, DeclParams, _Perms),
    ( Clause = (Head :- Body) ->
        true
    | otherwise ->
        Head = Clause,
        Body = true
    ),
    append(Params, DeclParams, AllParams),
    ( memberchk(callback(Callback), AllParams) -> true | true ),

    ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
      ( icl_GetParamValue(owner(Owner), Params) -> true | true ),
        ( icl_GetParamValue(do_all(true), Params) ->
            oaa_retractall(Clause, Owner, Callback)
      | otherwise ->
          oaa_retract(Clause, Owner, Callback)
      )
    | otherwise ->
        ( icl_GetParamValue(do_all(true), Params) ->
            retract_all(Clause)
      | otherwise ->
            retract(Clause)
      )
    ),

    oaa_CheckTriggers(data, Head, remove),
    !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:       oaa_replace_data_local(+Clause1, +Params)
% purpose:    Replace one or more clauses from an agent's solvable.
% arguments: See comments for oaa_ReplaceData.
```

```
% remarks:
%   This performs the local processing needed for calls to oaa_ReplaceData, and
%   ev_update(replace, ...) requests.
%   Clause1 is the thing to be replaced.  The thing to replace it with must
%   be present in Params, as with(Clause2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_replace_data_local(Clause1In, Params) :-
    memberchk(with(Clause2In), Params),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_data_matches_solvables(Clause1In, Solvables, write, Clause1, Matched),
    oaa_data_matches_solvables(Clause2In, Solvables, write, Clause2, _Matched2),
    Matched = solvable(_Pred, DeclParams, _Perms),
    ( Clause1 = (Head :- Body) ->
        true
    | otherwise ->
        Head = Clause1,
        Body = true
    ),

    append(Params, DeclParams, AllParams),
    ( memberchk(callback(Callback), AllParams) -> true | true ),

    % do replace of either one or all occurrences
    ( \+ icl_GetParamValue(bookkeeping(false), DeclParams) ->
        oaa_data_owner(Params, Owner),
        ( icl_GetParamValue(do_all(true), Params) ->
            oaa_replace_all(Clause1, Clause2, Owner, Callback)
        | otherwise ->
            oaa_retract(Clause1, _OldOwner, Callback),
            (icl_GetParamValue(at_beginning(true), AllParams) ->
               oaa_asserta(Clause2, Owner, Callback)
          |  oaa_assertz(Clause2, Owner, Callback)
          )
      )
    | otherwise ->
        ( icl_GetParamValue(do_all(true), Params) ->
            replace_all(Clause1, Clause2)
      | otherwise ->
            retract(Clause1),
            (icl_GetParamValue(at_beginning(true), AllParams) ->
               asserta(Clause2)
          |  assertz(Clause2)
          )
      )
    ),
    oaa_CheckTriggers(data, Clause1, remove),
    oaa_CheckTriggers(data, Clause2, add),
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    retract_all
% purpose: Remove all clauses matching Clause1
% remarks: Always succeeds.  Needed because retractall((func(X) :- Y)) doesn't
%          work.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
retract_all(Clause1) :-
   retract(Clause1),
```

```
   fail.
retract_all(_Clause1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    replace_all
% purpose: Replace all clauses matching Clause1 by Clause2
% remarks: Always succeeds
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
replace_all(Clause1, Clause2) :-
   retract(Clause1),
   assert(Clause2),
   fail.
replace_all(_Clause1, _Clause2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_data_owner(+Params, -Owner)
% purpose: Determine data ownership from the available params
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_data_owner(Params, Owner) :-
    ( memberchk(owner(Owner), Params) ->
        true
    | memberchk(from(Owner), Params) ->
      true
    | oaa_Id(Owner) ->
      true
    | otherwise ->
      Owner = unknown
    ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_Id(MyId)
% purpose: Return the Id of the current agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if connected to a Facilitator, use this Id
oaa_Id(MyId) :-
      com:com_GetInfo(parent, oaa_id(MyId)), !.
% For root, get any id
oaa_Id(MyId) :-
      com:com_GetInfo(ConnectionId, type(server)),
      com:com_GetInfo(ConnectionId, oaa_id(MyId)), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_Name(MyName)
% purpose: Return the name of the current agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if connected to a Facilitator, use this Id
oaa_Name(MyName) :-
      com:com_GetInfo(parent, oaa_name(MyName)), !.
% For root, get any id
oaa_Name(MyName) :-
      com:com_GetInfo(ConnectionId, type(server)),
      com:com_GetInfo(ConnectionId, oaa_name(MyName)), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_class(MyClass)
```

```
% purpose: Return the class (leaf, node, root) of the current agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% if connected to a Facilitator, use this Id
oaa_class(leaf) :-
      com:com_GetInfo(_, type(client)),
      \+ com:com_GetInfo(_, type(server)), !.
oaa_class(node) :-
      com:com_GetInfo(_, type(client)),
      com:com_GetInfo(_, type(server)), !.
oaa_class(root) :-
      com:com_GetInfo(_, type(server)),
      \+ com:com_GetInfo(_, type(client)), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name: oaa_asserta(Clause, Owner, SpecifiedCallback)
%       oaa_assertz(Clause, Owner, SpecifiedCallback)
%       oaa_retract(Clause, Owner, SpecifiedCallback)
%       oaa_retractall(Clause, Owner, SpecifiedCallback)
%       oaa_replace_all(Clause1, Clause2, Owner, SpecifiedCallback)
% purpose: Perform data updates with bookkeeping info (in oaa_data_ref/3)
% remarks: These should only be used with data solvables having param
%          bookkeeping(true).
%          There are still a couple limitations related to data callbacks.
%          First, callbacks don't work when bookkeeping(false).
%          Second, oaa_replace_all assumes the same callback is appropriate
%          for both the old and new facts.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_asserta(Clause, Owner, Callback) :-
    asserta(Clause, Ref),
    now(Time),
    assert(oaa_data_ref(Ref, Owner, Time)),
    oaa_call_callback(app_on_data_change, Callback, [add(Clause)]).

oaa_assertz(Clause, Owner, Callback) :-
    assertz(Clause, Ref),
    now(Time),
    assert(oaa_data_ref(Ref, Owner, Time)),
    oaa_call_callback(app_on_data_change, Callback, [add(Clause)]).

oaa_retract(Clause, Owner, Callback) :-
    ( Clause = (Head :- Body) ->
        true
    | otherwise ->
        Head = Clause,
        Body = true
    ),
    clause(Head, Body, Ref),
    ( retract(oaa_data_ref(Ref, Owner, _)) ->
        erase(Ref),
      oaa_call_callback(app_on_data_change, Callback, [remove(Clause)])
    ).

oaa_retractall(Clause, Owner, Callback) :-
    ( Clause = (Head :- Body) ->
        true
    | otherwise ->
```

```
        Head = Clause,
        Body = true
    ),
    clause(Head, Body, Ref),
    ( retract(oaa_data_ref(Ref, Owner, _)) ->
        erase(Ref),
      oaa_call_callback(app_on_data_change, Callback, [remove(Clause)])
    ),
    fail.
oaa_retractall(_Clause, _Owner, _Callback).

oaa_replace_all(Clause1, Clause2, Owner, Callback) :-
   oaa_retract(Clause1, _OldOwner, Callback),
   oaa_assertz(Clause2, Owner, Callback),
   % This would be redundant:
   % oaa_call_callback(app_on_data_change, Callback, [replace(Clause1,
Clause2)]),
   fail.
oaa_replace_all(_Clause1, _Clause2, _Owner, _Callback).



%**************************************************************************
% Trigger Handling
%**************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_CheckTriggers
% purpose: Given a trigger type, a mask and an Op (e.g. [send, receive],
%   [add, remove], etc), see if any triggers fire.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_CheckTriggers(Type, Condition, Op) :-
      % for each matching trigger
      oaa_solve_local(
            oaa_trigger(TriggerId, Type, Condition, Action, Params),
          []),

      ( (Type == task, \+ var(Condition)) ->
         % We don't want this to succeed more than once, so use ->
         ( oaa_Interpret(Condition, [from(self)]) -> true )
      | otherwise ->
          true
      ),

      % see if on(Op) has been specified
      (memberchk(on(OpSpecified), Params) ->
         OpMask = OpSpecified
      |  OpMask = _),

      % see if Op is OK
      ( (ground(OpMask), OpMask = [_|_]) ->
          memberchk(Op, OpMask)
      | otherwise ->
            Op = OpMask
      ),

      % test additional conditions
```

```
      (memberchk(test(Test), Params) ->
         % We don't want this to succeed more than once, so use ->
         ( oaa_Interpret(Test, [from(self)]) -> true )
      |  Test = 'true'),

      % check recurrence: remove trigger?
      (remove_element(recurrence(R), Params, NewParams) ->
         (R = whenever ->
            true        % don't remove trigger if 'whenever'
         | integer(R), R > 1 ->
            R2 is R - 1,
            % decrement recurrence count
            oaa_remove_data_local(
                oaa_trigger(TriggerId, Type, Condition, Action, Params),
              []),
            oaa_add_data_local(
                  oaa_trigger(TriggerId, Type, Condition, Action,
                           [recurrence(R2)|NewParams]),
                []))
         | oaa_remove_local_trigger_by_id(TriggerId)
         )

      |
         R = when,
         oaa_remove_local_trigger_by_id(TriggerId)
      ),

      oaa_TraceMsg(
       '~n~q trigger fired (~q): ~q AND ~q,~n  Action: ~q~n',
         [Type, Op, Cond, Test, Action]),

      (Type \== comm ->
         oaa_Inform(trigger,
            'trigger_fired(~q,~q,~q,~q)~n',
             [Type, Cond, Action, Params])
      | true),

      % FIRE!!!!
        oaa_fire_trigger(Action),

      % loop back for more triggers
      fail.
oaa_CheckTriggers(_Type, _Cond, _Op).

oaa_fire_trigger(oaa_Solve(Goal, Params)) :-
    !,
    ( memberchk(block(_), Params) ->
      NewParams = Params
    | otherwise ->
      append([block(false)], Params, NewParams)
    ),
    oaa_Solve(Goal, NewParams).
oaa_fire_trigger(oaa_Solve(Goal)) :-
    !,
    oaa_Solve(Goal, [block(false)]).
oaa_fire_trigger(oaa_Interpret(Goal, Params)) :-
    !,
```

```
    ( memberchk(from(_), Params) ->
      NewParams = Params
    | otherwise ->
      oaa_Id(Me),
      append([from(Me)], Params, NewParams)
    ),
    oaa_Interpret(Goal, NewParams).
oaa_fire_trigger(oaa_Interpret(Goal)) :-
    !,
    oaa_Id(Me),
    oaa_Interpret(Goal, [from(Me)]).
oaa_fire_trigger(Goal) :-
    oaa_Id(Me),
    oaa_Interpret(Goal, [from(Me)]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_AddTrigger
% purpose: Adds a trigger according to parameters
% Type     = comm, data, task, time
% Condition= comm:event to match,  data:data to match, task:solvable to call
%            time:@@
% Action   = Can be any of these:
%              oaa_Solve(Goal, Params)
%              oaa_Interpret(Goal, Params)
%              Goal [passed to oaa_Interpret with default params]
% Params   =
%   address(X): a list including 'self', 'parent', and/or the
%     addresses of other client agents.  Default: see below.
%   test(T): additional tests before trigger will fire [@@needs work?]
%   on(OP) : operation check: on(add), on(remove), on(receive), etc.
%   recurrence(R): when, whenever, or integer (# of times to execute)
%   reply({true,none}): When a trigger is being added on
%     a remote agent or agents, this tells whether reply message(s) are
%     desired.
%   block(Mode) : true: Block until the reply arrives.
%                 : false: Don't block.  In
%                          this case, the reply events
%                           can be handled by the user's app_do_event callback
%                 Default: true.  Note that reply(none) overrides
%                          block(true).
%   get_address(X): Returns a list of addresses (ids) of agents that
%     were sent the request.
%   get_satisfiers(X): Returns a list of addresses (ids) of agents that
%     successfully completed the request.
%
% Default destination for triggers:
%    Data triggers: all agents with solvables matching the Condition
%        field.
%    All other types: the local agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddTrigger(Type, Condition, Action, InitialParams) :-
   oaa_update_trigger(add, Type, Condition, Action, InitialParams).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_RemoveTrigger
```

```
% purpose: Removes a trigger from a local or remote agent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_RemoveTrigger(Type,Condition,Action,Params) :-
   oaa_update_trigger(remove, Type, Condition, Action, Params).

oaa_update_trigger(Mode, Type, InCondition, Action, InParams) :-
   ( (Type == comm, \+ InCondition = event(_,_)) ->
       Condition = event(InCondition, _)
   | otherwise ->
       Condition = InCondition
   ),
   icl_standardize_params(InParams, false, Params),
   % Is there a specified address?
   ( memberchk(address(Addr), Params) ->
       true
   | otherwise ->
       Addr = []
   ),

   % Decide whether or not to update locally:
   oaa_Id(Me),
   ( Addr \== [], memberchk(Me, Addr) ->
       delete(Addr, Me, NewAddr),
       replace_element(address(Addr), Params, address(NewAddr), Params1),
       Self = true
   | Addr = [], Type == data, icl_GetParamValue(reflexive(true), Params) ->
       % Do NOT use remove_element here:
       delete(Params, reflexive(true), Params1),
       NewAddr = Addr,
       Self = true
   | Addr = [], Type \== data ->
       NewAddr = Addr,
       Params1 = Params,
       Self = true
   | otherwise ->
       NewAddr = Addr,
       Params1 = Params
   ),

   % Update locally if appropriate:
   ( Self == true ->
       Requestees1 = [Me],
       ( Type == add ->
           Functor = oaa_add_trigger_local
       | otherwise ->
           Functor = oaa_remove_trigger_local
       ),
       LocalCall =.. [Functor, Type, Condition, Action, Params1],
       ( call(LocalCall) ->
           Updaters1 = [Me]
       |   Updaters1 = [])
   | otherwise ->
       Requestees1 = [],
       Updaters1 = []
   ),

   % Update remotely if appropriate:
```

```
    ( oaa_class(leaf), ((Addr == [], Type = data) ; NewAddr \== []) ->
        % Send the request event to the Facilitator
        oaa_PostEvent(
          ev_post_trigger_update(Mode,Type,Condition,Action,Params1), []),
        ( (icl_GetParamValue(reply(asynchronous), Params) ;
           icl_GetParamValue(reply(none), Params)) ->
            Requestees2 = [],
            Updaters2 = []
        | otherwise ->
            % In the return event, Requestees lists all agents to whom
            % the update request was sent; Updaters2 lists those who succeeded.
            oaa_poll_until_event(
              ev_reply_trigger_updated(Mode, Type, Condition, Action, Params1,
                                       Requestees2, Updaters2))
        )
    | otherwise ->
        Requestees2 = [],
        Updaters2 = []
    ),
    append(Updaters1, Updaters2, Updaters),
    % Return Updaters if requested:
    ( memberchk(get_satisfiers(Updaters), Params) -> true | true ),
    append(Requestees1, Requestees2, Requestees),
    % Return Requestees if requested:
    ( memberchk(get_address(Requestees), Params) -> true | true ).

oaa_add_trigger_local(Type, Condition, Action, Params) :-
    gensym(trg, TriggerId),
    oaa_add_data_local(
        oaa_trigger(TriggerId, Type, Condition, Action, Params),
      []).

oaa_remove_trigger_local(Type, Condition, Action, Params) :-
    oaa_remove_data_local(
        oaa_trigger(_TriggerId, Type, Condition, Action, Params),
      []).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_remove_local_trigger_by_id
% purpose: Removes a local trigger given its unique identifier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_remove_local_trigger_by_id(TriggerId) :-
     oaa_remove_data_local(oaa_trigger(TriggerId, _,_,_,_), []),
     !.



%*************************************************************************
% Requesting Services
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Solve
% purpose: Sends work or information requests to distributed agents, brokered
%        by the Facilitator agent
```

```
%
%   The default behavior (paramlist = []) is to act like the Prolog primitive
%   call(Goal), blocking until Goal is finished, and unifying and backtracking
%   over solutions for Goal.
%
%   This behavior may be modified by a parameter list, which may contain:
%
%       cache(T_F)          : cache all solutions locally, and if good solutions
%                         already exist in the cache, use the local values
%                         instead of making a distributed request.
%                               Default: false.
%       level_limit(N)    : highest number of hierarchical levels to climb for
%                               solutions.
%       address(AgentId): send request to specific agent, given its name or Addr
%                     If AgentID is 'self', solves the goal locally
%       reply(Mode)         : true: Reply desired.
%                           : none: No reply desired.
%                           Default: true, except when the call to oaa_Solve
%                             is a trigger action, in which case it is
%                             none.  'none' is used here instead of false,
%                             because we anticipate some additional values.
%       block(Mode)         : true: Block until the reply arrives.
%                           : false: Don't block.  In
%                             this case, the reply events (ev_reply_solved)
%                             can be handled by the user's app_do_event callback
%                           Default: true, except when the call to oaa_Solve
%                             is a trigger action, in which case it is
%                             false.  Note that reply(none) overrides
%                             block(true).
%       solution_limit(N)
%                    : limits the maximum number of solutions found to N
%       time_limit(N)     : Waits a maximum of N seconds before returning
%                         (failure if no solution found in time).
%       context(C)          : Passes a context value through any subsequent
%                     solves.
%       parallel_ok(T_F): if T_F is 'true' (default), multiple agents
%                     that can solve the Goal will attempt to work on it
%                     in parallel.  If 'false', one agent will be selected
%                     at a time to solve the goal, until the maximum
%                     number of requested solutions (see solution_limit) is
%                     found.
%     reflexive(T_F)
%                  : If T_F is `true', the Facilitator will consider the
%                       originating agent when choosing agents to solve a
%                       request.  Default: true.
%     priority(P) :  P ranges from 1 (low priority) to 10 (high priority)
%                               with a default of 5.
%       flush_events(T_F)
%                           : Will flush (dispose of) all events of lower priority
%                     currently queued at the destination agent.  These
%                     events are lost, and will not be executed.
%                     This parameter should be used with caution!!!
%                           Default: false.
%       get_address(X)    : Returns a list of addresses (ids) of agents that
%                           were asked to solve the goal, or one of its subgoals
%       get_satisfiers(X)
%                           : Returns a list of addresses (ids) of agents that
```

```
%                          succeeded in solving the goal, or one of its
%                          subgoals.
%
%       strategy(S)      : Shorthand for certain combinations of the above
%                          parameters.  S is one of
%                            query = [parallel_ok(true)]
%                            action = [parallel_ok(false), solution_limit(1)]
%                            inform = [parallel_ok(true), reply(none)]
%
%   Remarks: Note that certain combinations of parameters are inconsistent,
%   and are handled as follows:
%       reply(none) overrides block(true)
%       reply(none) overrides parallel_ok(false)
%
%   All of the above parameters may be used in the "global" parameter
%   list (the second argument to oaa_Solve), when Goal is non-compound.
%   Most can be used in the global list with compound goals also.
%   Some of these parameters can also be used in the NESTED parameter
%   lists of compound goals.  Uses of these parameters with compound
%   goals are documented elsewhere.  When that documentation exists,
%   this will go there:
%   With many compound goals, however, the get_satisfier/1 parameter isn't
%   really meaningful.  Thus, with compound goals, it is often best to use
%   this parameter in a nested parameter list.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Solve(Goal, InitialParams) :-
      % Trace message
      oaa_TraceMsg('~n~nStarting oaa_Solve request:~n   ~q [~q]...~n',
         [Goal,Params]),

      icl_standardize_params(InitialParams, false, Params),
      % Check for inappropriate params
        ( icl_GetParamValue(cache(true), Params), icl_compound_goal(Goal) ->
          format('~w: ~w  (~w)~n  Goal: ~w~n',
               ['WARNING', 'Ignoring ''cache'' parameter',
                'cannot be used with compound goal', Goal]),
            Compound = true
      | otherwise ->
          Compound = false
      ),

      % Add context to params
      ( oaa_current_contexts(_, Contexts) ->
          append(Contexts, Params, NewParams)
      | otherwise ->
          NewParams = Params
      ),
          ,
      % check cache
      (icl_GetParamValue(cache(true), NewParams), \+ Compound,
       on_exception(_, oaa_InCache(Goal, Solutions), fail) ->
           oaa_TraceMsg('~n~nSolutions found in cache:~n   ~q.~n',
              [Solutions])
      |
          % Should I solve this only locally?
        (oaa_Id(Me),
```

```
          memberchk(address(Me), Params) ->
           findall(Goal, oaa_solve_local(Goal, NewParams), Solutions)

       |
         % send request to Facilitator
         oaa_cont_solve(Goal, NewParams, Solutions),

         % print appropriate trace message
         (icl_GetParamValue(reply(none), NewParams) ->
            oaa_TraceMsg('~n~nMessage broadcast.~n', [])
         |
            oaa_TraceMsg('~n~nSolutions returned:~n   ~q.~n',
              [Solutions])
         ),

         % cache returned solutions if necessary
         ((icl_GetParamValue(cache(true), NewParams), Solutions \== []) ->
            oaa_AddToCache(Goal, Solutions),
            oaa_TraceMsg('Solutions cached.~n',[])
         | true)
       )

     ),!,

     % backtrack over all solutions
     member(Goal, Solutions).

oaa_solve_local(FullGoal, Params) :-
    % Validate the goal:
    icl_GoalComponents(FullGoal, _, Goal1, GoalParams),
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    ( icl_compound_goal(Goal1) ;
      icl_BuiltIn(Goal1) ;
      oaa_goal_matches_solvables(Goal1, Solvables, Goal, Matched) ),
    !,

    % More "local" params take precedence, so they go to the
    % beginning of the list:
    append([GoalParams, Params], AllParams),

    % We don't want tests to be performed repeatedly with compound goals,
    % so we remove them after testing.
    ( passes_tests(AllParams) ->
        delete(AllParams, test(_), NewParams),
      ( ( \+ var(Matched), Matched = solvable(_, SolvParams, _),
          icl_GetParamValue(type(data), SolvParams) ) ->
            ( memberchk(solution_limit(N), AllParams) ->
                call_n(N, Goal)
            | otherwise ->
                call(Goal)
            )
      | otherwise ->
          ( memberchk(solution_limit(N), AllParams) ->
              call_n(N, oaa_Interpret(Goal, NewParams))
          | otherwise ->
              oaa_Interpret(Goal, NewParams)
          )
```

```
      )
    | otherwise ->
        oaa_TraceMsg('~nDoesn''t pass test in: ~q~n', [AllParams]),
      fail
    ).

oaa_solve_local(FullGoal, _Params) :-
        format('~nError: do not know how to solve: ~q~n', [FullGoal]), fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_cont_solve
% purpose: Post request for solutions, and if appropriate, poll until
%          results are returned.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_cont_solve(Goal, GlobalParams, Solutions) :-
      % Send the ev_post_solve event to the Facilitator
      oaa_PostEvent(ev_post_solve(Goal, GlobalParams),[]),

      % Compound goals may also contain relevant params
        icl_GoalComponents(Goal, _, _, Params),
      append(Params, GlobalParams, AllParams),

      % If delayed reply or no reply OK, succeed immediately
      ( ( icl_GetParamValue(reply(false), AllParams) ;
          icl_GetParamValue(reply(none), AllParams) ;
          icl_GetParamValue(block(false), AllParams) ) ->
         Solutions = [Goal],
         Requestees = [],
         Solvers = []
      |
         % otherwise wait for solutions to return

         icl_GetParamValue(priority(P), AllParams),
         oaa_poll_until_event(ev_reply_solved(Requestees, Solvers, Goal,
SolvedParams, Solutions),
                              P),
        % The facilitator is responsible for making SolvedParams
        % unifiable with GlobalParams.  This msg is to keep facilitator
        % writers honest.
        ( GlobalParams = SolvedParams ->
            true
        | otherwise ->
            format('~w: ~w ~w~n  ~w: ~w~n',
                 ['WARNING:', 'Params in solved event don''t unify',
                  'with original params', 'SolvedParams', SolvedParams])
        )
      ),
        % Return Solvers if requested:
      ( memberchk(get_satisfiers(Solvers), GlobalParams) -> true | true ),
        % Return Requestees if requested:
      ( memberchk(get_address(Requestees), GlobalParams) -> true | true ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Solve/1
% purpose: Convenience function:  oaa_Solve with default parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
oaa_Solve(Goal) :- oaa_Solve(Goal, []).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_InCache
% purpose: Retrieve solutions from the cache if the goal we are
%          asking for is properly contained in the cache (check subsumption)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_InCache(Goal, Solutions):-
    oaa_cache(SomeGoal, _),
    subsumes_chk(SomeGoal, Goal),
    !,
    findall(Solution, oaa_cache(Goal, Solution), Solutions).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_AddToCache
% purpose: Add each solution to goal one at a time
%        so we can retrieve solutions later using findall
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_AddToCache(Goal, Solutions) :-
        member(Solution, Solutions),
         \+ oaa_cache(Goal, Solution),
        assert(oaa_cache(Goal, Solution)),
        fail.
oaa_AddToCache(_Goal, _Solutions).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_ClearCache
% purpose: Clear the cache
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ClearCache :-
    retractall(oaa_cache(_,_)).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_poll_until_event
% purpose: Block until requested event arrives in oaa_GetEvent
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_poll_until_event(Event) :-
    icl_param_default(priority(P)),
    oaa_poll_until_event(Event,P).

oaa_poll_until_event(Event,Priority) :-
      oaa_poll_until_all_events([Event],Priority).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_poll_until_all_events
% purpose: Block until all requested events arrive
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% no more events: we're done!
```

```
oaa_poll_until_all_events([], _Priority) :- !.

%% @@Adam - you were apparently working on this; I corrected a syntax
%% error or two, but otherwise left it alone.  - Dave
oaa_poll_until_all_events(EventList, Priority) :-
      % If we have a waiting_event, grab it
      %   see problem description in (oaa_is_waiting_for)
      (oaa_grab_waiting_event(EventList, Event) ;
       oaa_GetEvent(Event, Params, 0)),

      % if timeout returned, check triggers and call user:oaa_AppIdle
      %    then fail (continue with next clause)
      (Event = timeout ->
         oaa_CheckTriggers(task, _, _),
         oaa_call_callback(app_idle, _, []),
         fail
      |
         oaa_cont_poll_until_all_events(EventList, Event, Params, Priority)
      ), !.

% if oaa_GetEvent fails (e.g. timeout), just continue waiting
oaa_poll_until_all_events(EventList, Priority) :-
      oaa_poll_until_all_events(EventList, Priority).

oaa_cont_poll_until_all_events(EventList, Event, _Params, Priority) :-
      remove_element(Event, EventList, NewEventList), !,
      oaa_poll_until_all_events(NewEventList, Priority).
oaa_cont_poll_until_all_events(EventList, Event, Params, Priority) :-
      % if the new event is a ev_reply_solved() message for which we
      % are waiting at a higher recursive level, save this for
      % a later time, until we pop back out to the correct level.
      (oaa_is_waiting_for(Event) ->
         assert(oaa_waiting_event(Event))
      |
         % record what events we are waiting for on this processing level
         gensym(wait, WaitId),
         assert(oaa_waiting_for(WaitId, EventList)),

         (oaa_ProcessEvent(Event, Params) | true), !,

         % level over, remove waiting statement
         retract(oaa_waiting_for(WaitId, EventList))
      ),
      oaa_poll_until_all_events(EventList, Priority).

%*************************************************************************
% Callbacks
%*************************************************************************


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_RegisterCallback
% purpose: Declare what procedures should be used for callbacks.  These
%          are application-defined procedures called by library code.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

oaa_RegisterCallback(CallbackID, CallbackProc) :-
```

```
    ( CallbackProc = Module:Proc ->
      true
    | otherwise ->
      Module = user,
      Proc = CallbackProc
    ),
    retractall( oaa_callback(CallbackID, _) ),
    assert( oaa_callback(CallbackID, Module:Proc) ).

oaa_call_callback(CallbackID, SpecifiedCB, Args) :-
    ( ground(SpecifiedCB) ->
      SpecifiedCB = Module:Functor
    | otherwise ->
      oaa_callback(CallbackID, Module:Functor)
    ),
    !,
    Call =.. [Functor | Args],
    on_exception(E,
             Module:Call,
              ( oaa_TraceMsg('WARNING (oaa.pl): Exception raised thru callback
handler (~w):~n  ~q~n',
                             [Module:Functor, E]),
                fail )
            ).
oaa_call_callback(_CallbackID, _SpecifiedCB, _Args).

%*************************************************************************
% Debugging
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_TraceMsg
% purpose: If trace mode is on, display message and arguments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_TraceMsg(FormatString, Args) :-
      (oaa_trace(on) ->
          format(FormatString, Args)
%         oaa_Inform(trace_info, FormatString, Args)
          ;
          true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_ComTraceMsg
% purpose: If com trace mode is on, display message and arguments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_ComTraceMsg(FormatString, Args) :-
      (oaa_com_trace(on) ->
          format(FormatString, Args)
%         oaa_Inform(trace_info, FormatString, Args)
          ;
          true).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_turn_on_debug
% purpose: start debugging if debug mode is on
% remarks:
```

```
%    Use predicate_property and call so as to avoid errors in
%    building and running a Quintus runtime system.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_turn_on_debug :-
      (oaa_debug(on) ->
          ( predicate_property(user:trace, built_in) ->
              call(user:trace)
          | true )
        | true).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_turn_off_debug
% purpose: stop debugging if debug mode is on
% remarks:
%    Use predicate_property and call so as to avoid errors in
%    building and running a Quintus runtime system.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_turn_off_debug :-
      (oaa_debug(on) ->
          ( predicate_property(user:nodebug, built_in) ->
              call(user:nodebug)
          | true )
        | true).


%*************************************************************************
% User Interface
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_Inform
% purpose: sends a typed message to interested agents
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_Inform(TypeInfo, FormatString, Args) :-
          oaa_TraceMsg(FormatString, Args),
          (oaa_class(leaf) ->
             sprintf(Result, FormatString, Args),
             oaa_Solve(inform(TypeInfo, Result), [strategy(inform)])
            |
             true
          ), !.




%*************************************************************************
% Connection primitives
%*************************************************************************

%%% BUG/HACK!!!!!
% tcp_send/1 is not currently defined (new version of quintus)
% so these predicates should fail.  This means we can't have
% multilevel facilitators.
% However, if we fix it by the tcp_send/2 version (commented out),
% killing the agent doesn't shut down both connections and the
% facilitator server doesn't register the agent as disconnected.
```

```
% This must be fixed, but I don't have time now...

% Ask the root agent for the address of facilitator FacName.
% Either FacId or FacName may be bound.
% IMPORTANT: This assumes the root agent is the only connection when
% this is called.
% @@Not happy with the use of a Connection number in the address param here.
% Can an address be a connection number as well as an id or name???  [No.]

% get_address(FacId, FacName, Port, Host):-
%         tcp_connected(RootConnection),
%     oaa_Solve(agent_location(FacId, FacName, Port, Host),
%             [address(RootConnection)]).


%% succeed if FacName has not been registered with the root agent.
%%    otherwise, ask user to enter a different name for FacName

% check_name_duplication(MyName, NewMyName) :-
%         tcp_send(ev_check_agent_name(MyName)),
%         oaa_select_event(0, X),
%         oaa_extract_event(X, Result, _), %% 'UNIQUE'
%         (Result == 'UNIQUE' -> NewMyName = MyName
%                 ;
%         format('Name is duplicated~n',[]),
%         format('The following are registered ~n ~q ~n',[Result]),
%         format('Input agent name again:',[]),
%         read(NewMyName)).

% report_address_to_root(MyName, NewAddress):-
%     tcp_send(register_port_number(MyName, NewAddress)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% routines to fix bug:
%   blocking solve1
%      incoming event generates blocking solve2
%      solution to solve1 thrown away!!!
%      solutions to solve2
%   stuck  waiting for solve1 forever
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % name:    oaa_is_waiting_for
   % purpose: Check to see if the current event is something we are waiting
   %          for on a higher recursive level
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   oaa_is_waiting_for(Event) :-
        oaa_waiting_for(_Id, EventList),
        memberchk(Event, EventList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % name:    oaa_grab_waiting_event
   % purpose: If one of the delayed events is in the EventList that we are
   %     waiting for, return this event and remove from delayed list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   oaa_grab_waiting_event(EventList, Event) :-
```

```
        oaa_waiting_event(Event),
        memberchk(Event, EventList),
        !,
        retract(oaa_waiting_event(Event)).


%**************************************************************************
% OAA Utilities
%**************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_remove_solvables_data(Solvables).
% purpose: For each data solvable, remove all clauses belonging to it.
% remarks: - Solvables must be in standard form, and should include only
%            data solvables.
%          - Permissions are ignored.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_remove_solvables_data([]).
oaa_remove_solvables_data([Solvable | Solvables]) :-
    Solvable = solvable(Goal, Params, _Perms),
    icl_GetParamValue(type(data), Params),
    \+ memberchk(synonym(_, _), Params),
    !,
    % This should have already been done, but to be safe:
    (clause(Goal, _, _) -> true | true),
    predicate_skeleton(Goal, Skeleton),
    ( oaa_remove_data_local(Skeleton, [do_all(true)]) ->
        true
    | otherwise ->
        format('~w: Problem in removing all data for solvable: ~w~n',
             ['! ERROR', Goal])
    ),
    oaa_remove_solvables_data(Solvables).
oaa_remove_solvables_data([_Solvable | Solvables]) :-
    oaa_remove_solvables_data(Solvables).

oaa_remove_data_owned_by(Id) :-
    ( oaa_solvables(Solvables) -> true | otherwise -> Solvables = []),
    oaa_built_in_solvables(BuiltIns),
    append(BuiltIns, Solvables, AllSolvables),
    oaa_remove_data_owned_by(AllSolvables, Id).

oaa_remove_data_owned_by([], _Id).
oaa_remove_data_owned_by([Solvable | Solvables], Id) :-
    Solvable = solvable(Goal, Params, _Perms),
    icl_GetParamValue(type(data), Params),
    \+ icl_GetParamValue(persistent(true), Params),
    \+ icl_GetParamValue(synonym(_, _), Params),
    !,
    % This should have already been done, but to be safe:
    (clause(Goal, _, _) -> true | true),
    predicate_skeleton(Goal, Skeleton),
    ( oaa_remove_data_local(Skeleton, [owner(Id), do_all(true)]) ->
        true
    | otherwise ->
        format('~w: Problem in removing data owned by ~w for solvable:~n  ~w~n',
             ['! ERROR', Id, Goal])
```

```
    ),
    oaa_remove_data_owned_by(Solvables, Id).
oaa_remove_data_owned_by([_Solvable | Solvables], Id) :-
    oaa_remove_data_owned_by(Solvables, Id).


%*************************************************************************
% General Utilities
%*************************************************************************

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_consult(+FilePath, -AbsFileName).
% purpose:
% remarks: We don't use Quintus' builtin consult, because it's too picky
%          about associating predicates with files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_consult(FilePath, AbsFileName) :-
    absolute_file_name(FilePath, AbsFileName),
    can_open_file(AbsFileName, read, fail),
    open(AbsFileName, read, Stream),
    load_clauses(Stream),
    close(Stream).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clauses(+Stream).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clauses(Stream) :-
    repeat,
    read_term(Stream, [], Term),
    ( Term = ':-'(_Body) ->
      true
    | Term = end_of_file ->
      true
    | otherwise ->
        load_clause(Term)
    ),
    ( at_end_of_file(Stream) ->
      !
    | otherwise ->
      fail
    ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    load_clause(+Term).
% purpose:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load_clause(Term) :-
    assert( Term ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    oaa_declare_for_prolog(Solvables).
% purpose: For each solvable, make sure it's known to Prolog as a dynamic
%          predicate.  This will prevent exceptions and warnings from
```

```
%          calls and retracts before there have been any asserts.
% remarks: Solvables must be in standard form, and should include only
%          data solvables.
%          This is probably Quintus-specific.
%          We are assuming that none of these predicates are known to
%          Prolog as compiled predicates.  Would be better to check for this.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
oaa_declare_for_prolog([]).
oaa_declare_for_prolog([solvable(Pred, _, _) | Rest]) :-
    copy_term(Pred, PredCopy),
    ( clause(PredCopy, _Body) -> true | true ),
    oaa_declare_for_prolog(Rest).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    predicate_skeleton(+Goal, +Skeleton).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
predicate_skeleton(Goal, Skeleton) :-
    functor(Goal, Functor, Arity),
    functor(Skeleton, Functor, Arity).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    sprintf
% purpose: C-like command formats a string + args into an atom
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sprintf(AtomResult, FormatStr, Args) :-
   with_output_to_chars(format(FormatStr, Args), Chars),
   name(AtomResult, Chars).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    memberchk_nobind
% purpose: like memberchk, but doesn't bind variables in Elt when doing test.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
memberchk_nobind(Elt, [H|_]) :-
      would_unify(Elt, H), !.
memberchk_nobind(Elt, [_|T]) :-
      memberchk_nobind(Elt, T).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    would_unify
% purpose: succeeds if X and Y WOULD unify, but doesn't actually do the
%          unification (no variables are bound by test)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
would_unify(X,Y) :- \+ \+ X = Y.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    remove_element
% purpose: Removes the element X from a list
% remarks: Fails if X is not an element in the list
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
remove_element(X, [X|Rest], Rest) :- !.
remove_element(X, [Y|Rest], [Y|Rest2]) :- remove_element(X, Rest, Rest2).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    replace_element(Elt, List, New, NewList)
% purpose: Replaces the element Elt, if present in List, with the element New
% remarks: If there are multiple occurrences of Elt, only replaces the first
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
replace_element(Elt, [Elt|Rest], New, [New|Rest]) :- !.
replace_element(Elt, [Y|Rest], New, [Y|Rest2]) :-
    replace_element(Elt, Rest, New, Rest2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    select_elements(List, Selector, NewList)
% purpose: Selects all List elements for which Selector(element) succeeds.
% remarks: If there are multiple occurrences of Elt, only replaces the first
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
select_elements([], _Selector, []).
select_elements([Element | Elements], Selector, [Element | Selected]) :-
    Test =.. [Selector, Element],
    call( Test ),
    !,
    select_elements(Elements, Selector, Selected).
select_elements([_Element | Elements], Selector, Selected) :-
    select_elements(Elements, Selector, Selected).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% name:    call_n(+N, +Goal)
% purpose: Call Goal with a limit on the number of solutions generated.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

call_n(1, Goal) :-
    call(Goal),
    !.
call_n(N, Goal) :-
    % Remember the counter's value in case anyone else is using it.
    ctr_is(12, CtrOrig),
    call_n_aux(N, Goal, CtrOrig).

call_n_aux(N, Goal, CtrOrig) :-
    N > 1,
    ctr_set(12, 1),
    call(Goal),
    ctr_inc(12, 1, M),
    ( M =< N ->
        true
    | otherwise ->
        ctr_set(12, CtrOrig),
      !,
      fail
    ).
    % This clause is for when the Goal fails before M > N:
call_n_aux(_N, _Goal, CtrOrig) :-
    ctr_set(12, CtrOrig),
    !,
    fail.
```

```
% findall with a limit on the number of solutions generated.
findNSolutions(0, _Var, _Predicate, []).
findNSolutions(1, Var, Predicate, [Var]) :-
        call(Predicate), !.
findNSolutions(1, _Var, _Predicate, []).
findNSolutions(N, Var, Predicate, Solutions) :-
    N > 1,
    % Save the counter's value in case anyone else is using it.
    ctr_is(12, CtrOrig),
    ctr_set(12, 1),
    findall(Var,
            (Predicate, ctr_inc(12, 1, M),
             (M >= N -> ! | otherwise -> true)),
           Solutions),
    ctr_set(12, CtrOrig).

% ================================================================
% No longer used: replaced or obsolete
% ================================================================

% initialize all data flags
% oaa_init_flags :-
%     % set appropriate prolog flags
%     prolog_flag(fileerrors,_,on),
%     prolog_flag(syntax_errors,_,error),
%     % Let's use retractall so as to avoid unknown exceptions when tracing:
%     retractall(oaa_cache(_,_)),
%     retractall(oaa_already_loaded(_)),
%     assert(oaa_trace(off)),
%     assert(oaa_debug(off)),
%     assert(oaa_com_trace(off)),
%     tcp_trace(_,off).
```